Títol: Lemúria (Simulació d'un Ciutat Medieval)

Volum: 1

Alumne: Manel Rello Saltor

Director/Ponent: Pau Fonseca i Casas

Departament: EIO (Estadística i Investigació Operativa)

Dono gràcies a la Arquitectura perquè m'ha permès veure el món amb els seus ulls.

Rafael Moreno, Arquitecte

Si una inteligencia sobrehumana conegués per un instant totes les forces a les que està sotmesa la naturalesa i la posición relativa de tots els éssers que la componen, aleshores tindria el coneixement de l'estat present de l'univers com a efecte del seu estat anterior i causa del seu estat futur.

Pierre-Simon Laplace, Filòsof i Matemàtic

Índex de Continguts

1. Idea Inicial

La idea inicial d'aquest projecte era la creació d'una eina que permeti, a través de la simulació d'una ciutat, analitzar alguns dels seus aspectes permetent realitzar estudis sociològics.

Més en detall, el que es volia era crear una eina que pogués inferir el comportament que tindrien les persones d'una ciutat sota diverses condicions i esdeveniments per tal de poder-ne predir el comportament i actuar en conseqüència.

Exemples del possible funcionament de la eina seria la previsió del comportament de les persones i l'evolució del teixit de la ciutat sota un desastre natural, un atac militar o el comportament que tindrien en condicions de restriccions d'aigua o en l'arribada massiva d'immigració.

A simple vista i sense necessitat de coneixements d'informàtica es pot veure que una eina d'aquestes característiques és d'una mida considerable i que revesteix una gran dificultat al incorporar elements clarament en estadi de recerca. Aquestes dimensions la fan impossible de realitzar sota els terminis i les limitacions d'un projecte final de carrera convencional.

Amb l'objectiu de poder-ne realitzar una aproximació que sigui adequada a un projecte final de carrera, es va decidir d'intentar de limitar-ne l'abast a través d'una sèrie de simplificacions[1].

Una de les primeres simplificacions que es va convenir va ser la de simular una ciutat del passat. Amb això es guanyava en dos fronts, el primer era que el nombre de variables i la casuística que es donaven en una ciutat medieval són infinitament menors que els d'una ciutat actual. Començant pel fet de que el nombre de persones era molt menor, també el nombre d'ocupacions possibles era molt inferior i pràcticament la totalitat de la seva població es dedicava al cultiu dels camps o a la elaboració de productes. Això provoca que tot el sector

[1] Referir-se a la secció d'Hipòtesis Simplificadores (pàg 59)

de serveis es pot obviar en l'època medieval. La composició familiar i l'ocupació dels seus membres, a més, era també molt similar entre unes famílies i les altres i l'economia medieval, tot i que complexa, és molt més simple que l'actual. L'altre front on es guanya és que al ser una ciutat del passat, es disposa de dades que permeten la contrastació de l'evolució que proposa el simulador amb la que realment es va donar.

La segona simplificació que es va proposar va ser la de basar el simulador en les famílies enlloc de en individus. Això, que a priori pot semblar afegir un grau de complexitat enlloc d'un grau de simplificació permet reduir de forma considerable la dificultat del simulador. Les causes d'això parteixen, sobretot, del fet de que totes les relacions socials dels individus es poden obviar pràcticament en la seva totalitat (exceptuant les relacions mínimes per a poder crear parelles d'individus per a formar noves famílies). A més, al centralitzar la IA en la família fa que les reaccions dels membres d'una mateixa família siguin coherents entre elles sense necessitat d'establir comunicació entre elles.

2. Objectius del Projecte

Un cop s'havien establert les simplificacions principals que es farien a la idea inicial, es va obtenir una nova eina que sí que es podia adaptar a les limitacions d'un projecte final de carrera i que conservava encara la filosofia de la idea inicial.

Els nous objectius es van centrar en la construcció d'una eina **parametritzable** que permeti la simulació de qualsevol ciutat de l'Europa Occidental durant els S. XII i XIII.

Aquesta eina hauria de permetre l'estudi del comportament de les persones i l'evolució de la ciutat en els àmbits professional, demogràfic, econòmic i d'expansió de la ciutat.

En l'àmbit **professional**, la eina ha de ser capaç d'estudiar la orientació professional dels diferents individus de la ciutat així com les migracions internes entre els camps adjacents a aquesta i el nucli urbà.

El **creixement demogràfic** de la ciutat es centrarà en l'estudi de l'evolució de la quantitat de persones que viuen a la ciutat i a la seva àrea metropolitana.

En l'àmbit **econòmic**, hauria de permetre l'estudi de l'evolució de l'oferta i la demanda de productes així com l'evolució de preus i estocs dels diversos productes que formen l'economia de la ciutat.

L'estudi **migratori** consisteix en l'anàlisi dels fluxos migratoris des de la ciutat cap a altres terres o des de l'exterior de la ciutat cap al que seria l'àrea metropolitana.

La **Expansió de la ciutat** es centra en l'estudi de l'aparició o desaparició de noves vivendes, tallers, comerços dins del que seria el nucli urbà i observar-ne la tendència. Als segles que es simularan, aquest fet va ser un element realment important donat que es va produir una migració important cap a les ciutats i aquestes van créixer de forma considerable.

3. Anàlisi d'antecedents i factibilitat

3.1. Historia i Antecedents

L'objectiu final de la simulació és aconseguir reproduir en una màquina computacional el que es dona a la realitat. Des de l' "*Essai philosophique sur les probabilités*", obra de Pierre-Simon Laplace, que teoritzava amb el fet de que si es disposés de la informació suficient, un podria simular el TOT, un pot pensar que és possible que en un futur sigui possible arribar a simular amb bastant exactitud molts dels aspectes de l'univers. També existeixen algunes teories sobre si realment la nostra vida no transcorre dins d'un simulador controlat per un deu-informàtic...[2]

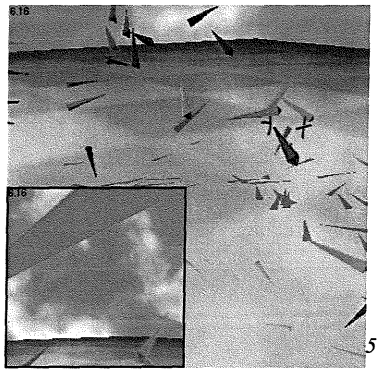
Tot i que la idea pot semblar temptadora, la irrupció del caos, el fet de que variant les condicions inicials en una proporció, els resultats variïn de forma imprevisible, en aquest aspecte fa impossible el plantejament de construir un simulador tan ambiciós.

No obstant, aspectes puntuals sí que es poden simular amb més o menys precisió i obtenint uns resultats que puguin servir per a l'estudi i la millora del camp estudiat.

Un dels punts que sempre ens ha temptat de simular és la vida. L'A-life, o vida artificial, és un dels camps que més ens criden l'atenció. Des de el primer *joc de la vida*, on els éssers vius vivien i es morien sobre una taula amb posicions fixes fins a les complexes simulacions d'ecosistemes i organismes que podem trobar en el *projecte Tierra*[3] o a la plataforma *Breve*[4].

[2] Veure http://www.simulation-argument.com/.

[3] *Segons Wikipedia*: Projecte desenvolupat per l'ecologista Thomas S. Ray a l'any 1990 en que els programes que corren sobre un ordinador competeixen entre ells per a obtenir els recursos (memòria, processador,...)



[5]

Però l'ésser humà sempre ha volgut anar a més i, el següent pas dins de l'àmbit de la simulació és simular-nos a nosaltres mateixos. Poder estudiar el comportament futur de les persones és un dels camps en que la simulació ha treballat profundament. Models per predir el comportament dels éssers humans en casos d'incendi, en les terminals d'aeroport o l'actitud que tindran quan entrin dins d'un tren que ve amb retràs, han estat alguns dels exemples de simulacions que s'han creat.

Però una de les idees que sempre han estat a la ment ha estat la simulació de ciutats completes. Evolució, moviment, compra, venda, relacions, tots aquests aspectes tan importants de la nostra vida dins d'un simulador capaç de predir com caurà l'oferta d'un producte determinat si es produeix un esdeveniment concret o quins seran els canvis socials sota condicions determinades.

S'ha d'especificar que existeixen alguns simuladors de ciutats, un dels punts on tenen més impacte és dins de l'àmbit de l'oci electrònic. Jocs com Sim City, els Sims, Black & White, Fable, Civilization,... han intentat de crear societats humanes simulades. No obstant, el seu objectiu és lúdic enlloc de científic, cosa que els fa poc adequats per a fer estudis sociològics.

---

[4] *Segons Wikipedia*: És un paquet de software que permet la creació senzilla de simulacions en 3D de forma distribuïda de vida artificial. (http://www.spiderland.org/breve/)

[5] Imatge dels resultat d'una simulació sota Breve

Tots ells, però, donat que tenen un objectiu lúdic i no científic, simulen el comportament de forma molt simple i molts cops condicionada a les accions del jugador. Per exemple, en el cas dels Sims, un simulador que s'anomena social en el que controlem a un ésser humà (anomenat sim) al llarg de la seva vida. El joc disposa d'una mini intel·ligència artificial que fa que si el jugador no pot controlar al seu personatge (perquè n'està controlant un altre, per exemple) aquest actuï de forma automàtica i sobrevisqui mínimament. No obstant, això no es pot considerar una simulació realista donat que les accions que pren el sim automàticament són molt limitades i si el jugador no intervé, morirà.

En l'àmbit dels simuladors científics de Ciutats, existeixen alguns projectes dins de l'àmbit universitari que intenten de simular ciutats. Universitats com la Universitat de Viena o el MIT han realitzat projectes de simulacions de ciutats però el que caracteritza el meu projecte és que al ser un simulador medieval, es poden contrastar els resultats obtinguts amb els que realment es van donar a la història.

3.2. Factibilitat dins del Marc del PFC

La creació d'un simulador d'una ciutat és un projecte molt ambiciós. No obstant, hi havia una sèrie d'elements que facilitaven que pogués ser factible la creació .

El **primer element** va ser que l'objectiu inicial no era crear un simulador plenament funcional sinó crear una estructura, un prototip primer, capaç de llençar simulacions de forma molt aproximada. Aquest primer punt va representar, a grosso modo, que el mida del projecte fos la de un pfc convencional.

El segon element que va propiciar la factibilitat va ser que es van aplicar una sèrie de simplificacions[6] que en reduïen encara més l'abast i, per tant, el convertien en un projecte de la mida adequada per a un pfc.

El tercer element va ser que ja es disposava d'una eina preparada per a poder llençar simulacions. El SDLPS, una plataforma per a poder llençar simulacions codificades en SDL[7] a través d'un XML de forma distribuïda i extraient-ne informació. L' SDLPS era una eina en fase de desenvolupament però que ja era pràcticament funcional. El fet de disposar ja d'una eina d'aquesta utilitat era un gran avantatge tot i que el fet d'utilitzar-la implicava haver de corregir els possibles errors o afegir funcionalitats que es podrien necessitar al llarg del desenvolupament.

En resum, aquests tres elements ajudaven a la realització del projecte i permetien que la seva mida el fessin factible tant a nivell de temps com a nivell d'objectius.

---

[6] Veure secció d'Hipòtesis Simplificadores (pàg 59)

[7] El SDL és un llenguatge de disseny que permet especificar models de simulació. Veure secció d'Introducció al SDL (pàg 21) per a més informació.

4. Estructura del Simulador

Tal com s'ha observat en les diverses fonts consultades sobre la societat medieval, aquesta girava en la seva majoria entorn a la producció dels diversos bens que existien. Aquesta producció els emprava la majoria de les hores del dia i de ella en depenien per a sobreviure.

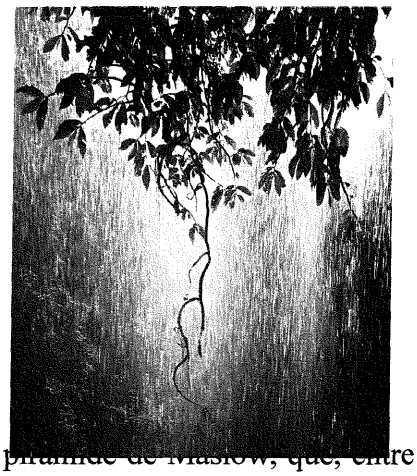
Això es deu, sobretot, al fet de que el menjar era un bé escàs i, per tant, es convertia en la seva prioritat al llarg de la seva vida.

També tenien relacions socials, tot i que menys extenses de les que podríem tenir ara pel fet de que la seva vida es centrava en aconseguir menjar[8].

Tenint en compte tota la informació recopilada sobre la ciutat medieval, el simulador s'ha dividit en 5 blocs funcionals que interactuen entre ells i que, d'aquesta forma, fan evolucionar tot el sistema. A continuació es descriu detalladament cadascun dels blocs i com es relaciona amb la resta d'elements.

4.1. Bloc de Clima

El bloc del clima, anomenat Enviroment als diagrames SDL[9], és el responsable de controlar el temps i el clima del simulador. Per una banda és el que envia senyals anuals i mensuals que marquen el pas del temps al sistema i, per



[8] Aquest fet s'explica amb facilitat a través de l'estudi de la piràmide de Maslow, que, entre d'altres, proposa que quan una persona no té menjar, les relacions socials estan en segon terme.

[9] Veure secció de Disseny SDL (pàg 28)

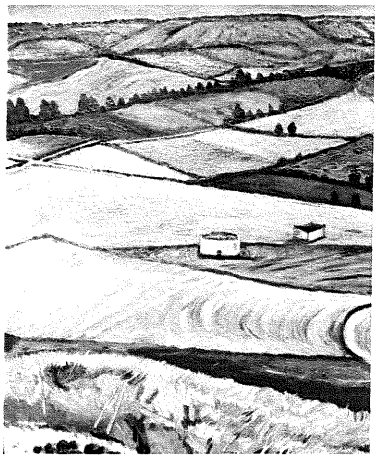
l'altra, general el clima que s'anirà succeint de forma mensual.

Al simulador, el clima es basa en tres elements principals: Pluja, Temperatura i Vent. A través d'aquests elements s'infereix quin és el clima que s'ha donat en l'últim més i s'actua en conseqüència. La pluja marca la quantitat d'aigua que ha caigut en l'últim mes, la temperatura és la mitjana de l'últim més i el vent és un indicador que marca la força amb la que ha bufat aquest.

Aquest bloc es relaciona amb la família i les propietats per a informar-los del temps que ha fet i amb el bloc del mercat per a donar-li informació temporal per a poder controlar l'evolució econòmica de forma ajustada.

4.2. Bloc de Propietats

Aquest bloc, anomenat Household al SDL[10], és el que s'encarrega de la simulació de les propietats de les famílies. És, per tant, l'encarregat de produir els recursos de que la família disposarà en funció de la família i el temps.



Existeixen dos tipus de propietats: les urbanes i les rurals. Les primeres són les que permeten la producció de productes agrícoles o ramaders (Sector primari) com el blat, les verdures o la carn. El segon grup és el que permet la producció de productes manufacturats (Sector secundari) com podria ésser la roba.

Donat que les diverses propietats es situen sobre un mapa concret, es permet que una es converteixi d'un tipus a l'altre per tal de permetre l'evolució de la mida de la ciutat.

La forma com una família pot modificar el comportament d'un camp es centra sobretot en el producte que hi vulguin cultivar / produir, en la dedicació que hi posin, en l'experiència que

[10] Veure secció de Disseny SDL (pàg 28)

tinguin realitzant aquest producte i en la qualitat de la propietat en qüestió per a la producció del producte en concret.

El clima afecta de forma contundent en les propietat, especialment en les agrícoles. Depenent del clima que faci, una propietat produirà més o menys productes, i, per tant, donarà més o menys recursos a la família que el treballi.

4.3. Bloc de Unitat Familiar

Aquest bloc és el bloc central del simulador. Als diagrames SDL[11] és referit com a Familiar Unit o com a UF.

Aquest bloc és el més complex donat que és el que té una primera aproximació a la Intel·ligència artificial i el que estableix relacions amb una varietat més gran de blocs.

Per una banda, la Unitat Familiar rep informació del bloc de Clima i del bloc de Propietats. Un li proporciona informació sobre l'estat del clima i l'altre li retorna els recursos que produeix a intervals mensuals.

Cada mes, després de rebre (o no) els recursos de les seves propietats, la Unitat familiar activa la seva IA[12] i genera una sèrie d'accions a realitzar per tal de seguir viva de la millor manera

[11] Veure secció de Disseny SDL (pàg 28)

[12] Per a més informació de la IA de la Unitat Familiar, referir-se a la secció de Decisions tècniques (pàg 61).

possible. Aquestes accions es comuniquen a les seves propietats, que actuaran d'acord amb allò que els marqui la Unitat Familiar a la que pertanyin. També interactuen amb el mercat, on vendran els seus excedents de productes i obtindran a canvi altres productes necessaris per a portar la millor vida possible.

De forma paral·lela, anualment la unitat familiar calcula el seu estat de salut i la possibilitat de que hi hagi hagut algun naixement o defunció al llarg de l'any. Si es dona el cas, el fet és notificat a la resta d'unitats familiars.

Quan un adult es queda vidu o un adolescent s'hi converteix, passa a ser casable. Quan això es dona, aquest individu es posa en contacte amb el bloc de relacions on mirarà de trobar una parella adient a la seva condició per a formar una nova unitat familiar o per a que aquesta nova persona s'afegeixi a la seva pròpia unitat familiar (en el cas d'una persona vídua que ja té fills i propietats).

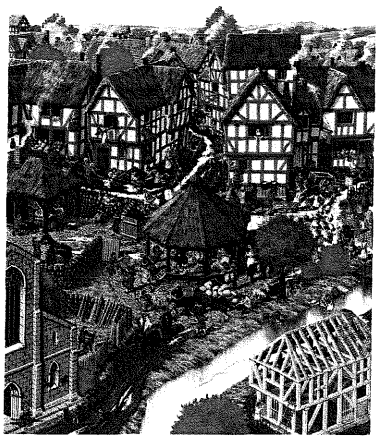
4.4. Bloc de Relacions

Aquest bloc, anomenat Couple Market als diagrames SDL[13], és el que s'encarrega de buscar parelles adients als individus adults sense parella. Quan hi ha un nou ingrés, es busca la parella adient per al nouvingut i, en cas de no trobar-la, aleshores s'espera a l'arribada d'un individu que acompleixi les expectatives.

Donat que una de les simplificacions del simulador era la de, precisament, fer la simulació a nivell de família per a no tenir la necessitat de controlar les relacions dels individus, aquest bloc és un dels més simples i la seva funció és merament la de trobar les parelles adients, cap d'altre.

[13] Veure secció de Disseny SDL (pàg 28)

4.5. Bloc de Mercat



El Mercat, anomenat Market als diagrames SDL[14], és el que s'encarrega de la compravenda dels productes. Per una banda rep productes que les famílies volen vendre i, per l'altra, rep peticions de productes de les famílies. Amb aquesta informació, el bloc analitza la situació i fa evolucionar els preus de forma coherent per tal de satisfer en la mesura del possible, les necessitats de les famílies que hi compren.

Un cop s'han efectuat les transaccions, el bloc envia a les famílies els productes que han adquirit a través del mercat.

Per tal de simplificar la feina sense entrar en detalls de moneda, s'ha creat un valor, anomenat UR (unitat de recursos) que s'utilitzarà per a marcar els preus dels diversos productes.

[14] Veure secció de Disseny SDL (pàg 28)

5. Disseny del Simulador

Com en la majoria de software que es desenvolupa, el disseny és la peça clau per a aconseguir uns bons resultats. En l'apartat anterior hem definit com s'estructuraria a nivell natural el simulador. En aquesta secció ens centrarem en el disseny tècnic del mateix partint de la idea no tècnica que ens hem formulat amb anterioritat.

5.1. Introducció al SDL

El primer de tot, però, és explicar quin mètode de disseny s'ha seguit. En aquest cas s'ha triat el SDL (Specification and Description Language [15]), un llenguatge utilitzat per a dissenyar sistemes distribuïts però que es pot aplicar i s'utilitza de forma bastant estesa en l'àmbit de la simulació.

Un dels avantatges que tindria l' SDL enfront d'altres llenguatge de disseny d software com l' UML és el fet de que té una forta component dinàmica. Implementa de forma natural i intrínseca que cadascun dels diversos elements que hi intervenen té un diagrama d'estats i que existeixen una sèrie de transicions i operacions entre ells. A més, és un llenguatge que permet distribuir el processat d'un mateix model de forma molt senzilla i pràcticament natural i que permet una integració directe en sistemes definits en UML..

A continuació s'especificaran quins són els elements bàsics que formen un disseny SDL. Aquesta explicació no pretén ser exhaustiva donat que seria massa extensa sinó que té l'objectiu de donar les claus bàsiques per a entendre el disseny del simulador .

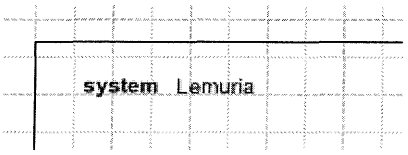
[15] Per a més informació, consultar

http://en.wikipedia.org/wiki/Specification_and_Description_Language i

http://www.sdl-forum.org/SDL/index.htm

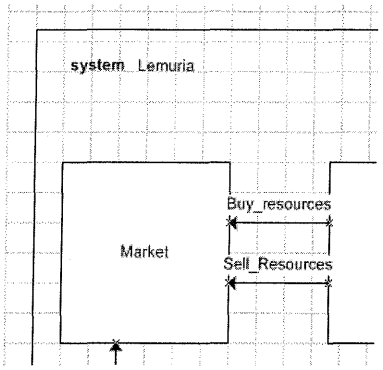
5.1.1. Diagrama de blocs

Tot diagrama SDL comença amb la creació d'un element anomenat **Sistema.** Aquest element representa TOT el nostre simulador i es dibuixa com un rectangle on hi anirem introduint la resta d'elements. A la part superior d'aquest element s'escriu la paraula System seguida del nom que tindrà el nostre simulador.



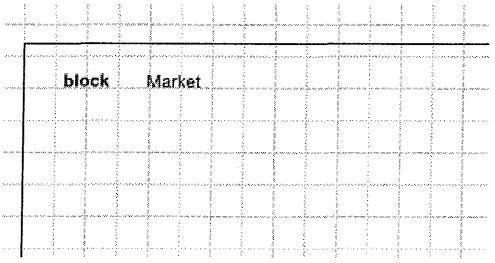
Dins del bloc de sistema podem introduir elements de dos tipus diferents: **Blocs** i **Processos.**

Els **blocs** són elements contenidors de forma similar al bloc de sistema. La única diferència que hi podem trobar és que d'ells hi poden sortir **canals** que els uneixen amb la resta de blocs i processos. De forma similar al bloc de sistema, els blocs poden contenir altres blocs i processos per tal de poder tenir un sistema més definit. Es representen com a quadrats amb el seu nom a l'interior en el diagrama de sistema o en el diagrama de blocs que els conté. A dins seu s'hi poden representar altres blocs o processos.

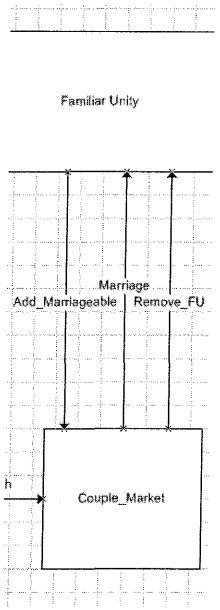


A la imatge anterior podem veure com dins del sistema s'ha definit un bloc anomenat Market que està connectat a altres blocs mitjançant una sèrie de canals.

Al seu torn, per raons d'espai i claredat, un bloc es pot representar de forma anàloga al sistema mitjançant un quadre en blanc amb la paraula clau del bloc a la part superior.



Un **canal** representa una unió entre blocs i processos per a permetre el pas d'informació codificada en forma de **senyals.** Aquestes senyals són la única via possible de comunicació entre els diversos blocs i processos i són el que desencadenarà el comportament i els canvis en els processos. La forma de representar un canal és mitjançant una línia, que pot ser una fletxa en el cas de que es vulgui especificar que el canal és en una direcció concreta, entre dos blocs o processos. Sobre el canal s'escriu el nom de la senyal o senyals que hi viatgen.



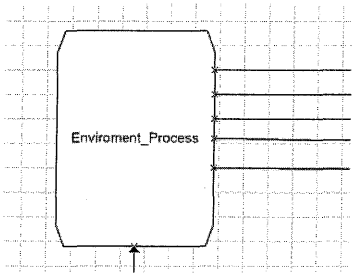
A la imatge anterior es poden veure dos blocs connectats per tres canals per on circulen les senyals de Add_Marriageable, Marriage i Remove_FU.

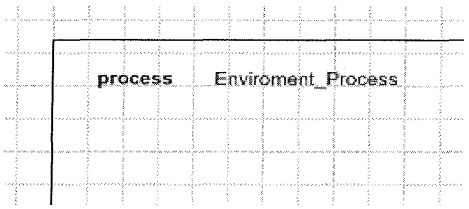
Un **procés** representa un element del simulador que no està composat per més. No obstant, dins d'un procés podrem definir més processos però al final definirà la màquina d'estats que defineix el comportament[16]. Dins seu hi ha definit un diagrama d'estats que en representa el

[16] La plataforma SDLPS no permet, de moment, descomposar un procés en més processos.

comportament dinàmic en funció de les senyals que li arriben a través dels canals que té connectats. A continuació s'explicarà amb més detall la forma d'aquests diagrames d'estats.

Un procés es representa amb un octàgon amb el nom del procés escrit a dins. De forma similar als blocs, quan es vol representar el contingut d'un procés es fa servir un quadre amb el nom del procés a la part superior esquerre precedit de la paraula "Process".





5.1.2. Diagrama d'estats

Tot procés té una sèrie d'estats associats. Aquests estats representaran en quina situació es troba el procés i quines han de ser les seves respostes en funció de les senyals que li arribin a través dels canals que té connectats. D'aquesta manera, per cada procés que forma el nostre simulador tindrem un diagrama d'estats associat.

Tot i semblar prou esclaridor, a la hora de representar el comportament, el diagrama d'estats és insuficient i necessitarem especificar a través d'un altra diagrama quines són exactament les operacions que s'han de seguir quan ens arriba una senyal determinada.

Per això, per cada estat definirem un diagrama que ens marcarà quines són les operacions que es fan i quines senyals s'envien com a resposta d'una senyal determinada.

Tot si semblar un diagrama important, normalment el diagrama d'estats s'obvia donat que la informació que proporciona és redundant si tenim en compte els diagrames de procés que es defineixen.

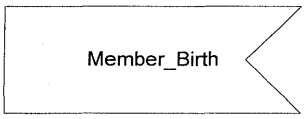
5.1.3. Diagrama de procés

Aquests diagrames parteixen de cadascun dels estats que formen el procés i en determinen una sèrie d'operacions. Les operacions que es poden realitzar com a conseqüència de l'arribada d'una senyal són molt variades i aquí en mostrarem les bases per a que sigui possible entendre el disseny del projecte.

Sempre es parteix de la representació de l'**estat**, que en aquest cas és una el·lipse o un rectangle amb els laterals arrodonits on s'hi inscriu el nom de l'estat.

D'aquesta figura (i de totes) en surt una línia que la uneix amb les següents operacions. D'aquesta forma, es defineixen una sèrie d'operacions de forma seqüencial que acaben sempre amb en un altre estat.

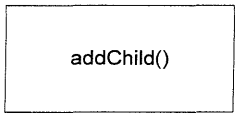
La segona operació en aquests diagrames serà sempre una de **l'arribada de senyal**. Aquesta operació serà el disparador de la que la segueixen en el cas de que la senyal representada arribi al procés. Es representa amb una rectangle on s'hi ha extret una part en forma de triangle a la dreta i s'hi inscriu el nom de la senyal que es vol que sigui el disparador.



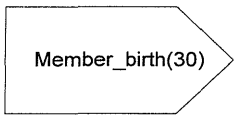
Quan ens arribi la senyal, efectuarem les operacions que segueixen a l'arribada d'aquesta senyal.

Una de les operacions que podem trobar a continuació és la **execució de codi**. Aquesta operació serà l'encarregada d'efectuar operacions diverses que tindrem codificades com a procediments en el SDL o algun llenguatge extern[17]. Tindran forma rectangular amb la crida a la funció o operacions a realitzar inscrites a l'interior.

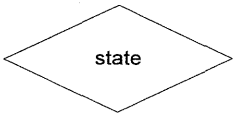
[17] En el nostre cas es va escollir la opció de codificar aquestes operacions en llenguatge C. Veure secció de Decisions tècniques (pag 61) decissions tècniques per a més información.



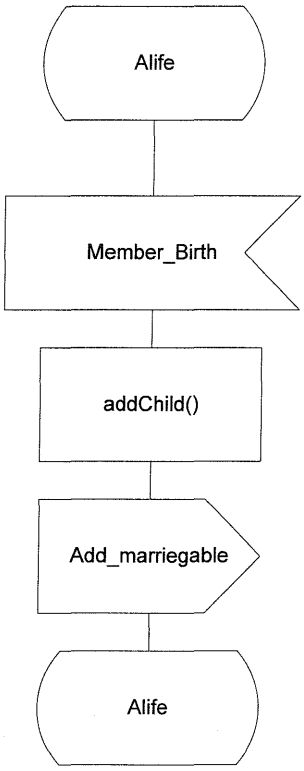
Una altra operació que podrem trobar és **l'enviament d'una senyal**. Aquestes es representen mitjançant un rectangle al que se li ha afegit un triangle a la part dreta. Aquestes operacions faran que la senyal especificada sigui enviada a través del canal corresponent. En cas de voler que una senyal s'enviï quan hagi passat un temps determinat, s'escriu aquest temps entre parèntesis després del nom de la senyal.



L'últim element utilitzat és el de **decisió**. Aquest especifica una condició que determinarà quin camí dels que parteixen d'aquesta operació hem de seguir. Es representa com un rombe on s'hi inscriu la condició. D'aquest rombe poden sortir diverses línies que l'uneixen amb diverses operacions. Cadascuna d'elles tindrà escrit a sobre el valor de la condició que farà que el flux segueixi per ells o no.



Un procés senzill podria tenir l'aparença següent:



En aquest procés es parteix de l'estat alife. Quan ens arriba la senyal de Member_birth, efectuem l'operació addChild() i enviem la senyal Add_marriegable. Després, tornem a l'estat Alife.

Tot procés sempre parteix d'un estat inicial que anomenarem estat 0. Per aquest estat només es pot passar una vegada, que és abans de començar la simulació. D'aquest estat només en sortirem un cop al inicial la simulació, i efectuarem les operacions que s'hi troben a continuació fins a situar-nos en un altre estat. Recordem que de la resta d'estats només se'n sortia amb l'arribada d'una senyal, d'aquest en sortirem de forma automàtica al començar la simulació.

L'estat 0 es representa com la resta d'estats amb la diferència de que no se li inscriu cap nom i la operació que vindrà a continuació de l'estat no serà mai la rebuda de senyal.

Com ja s'ha comentat, l' SDL és un llenguatge de gran complexitat i aquí només hem donat el subconjunt necessari per a poder entendre mínimament els diagrames que es mostraran a continuació.

Per a més informació referent al llenguatge SDL, podeu referir-vos a les següents adreces:

Web del SDL-forum[18]:

http://www.sdl-forum.org/

PDF amb una explicació extensa i detallada del llenguatge SDL:

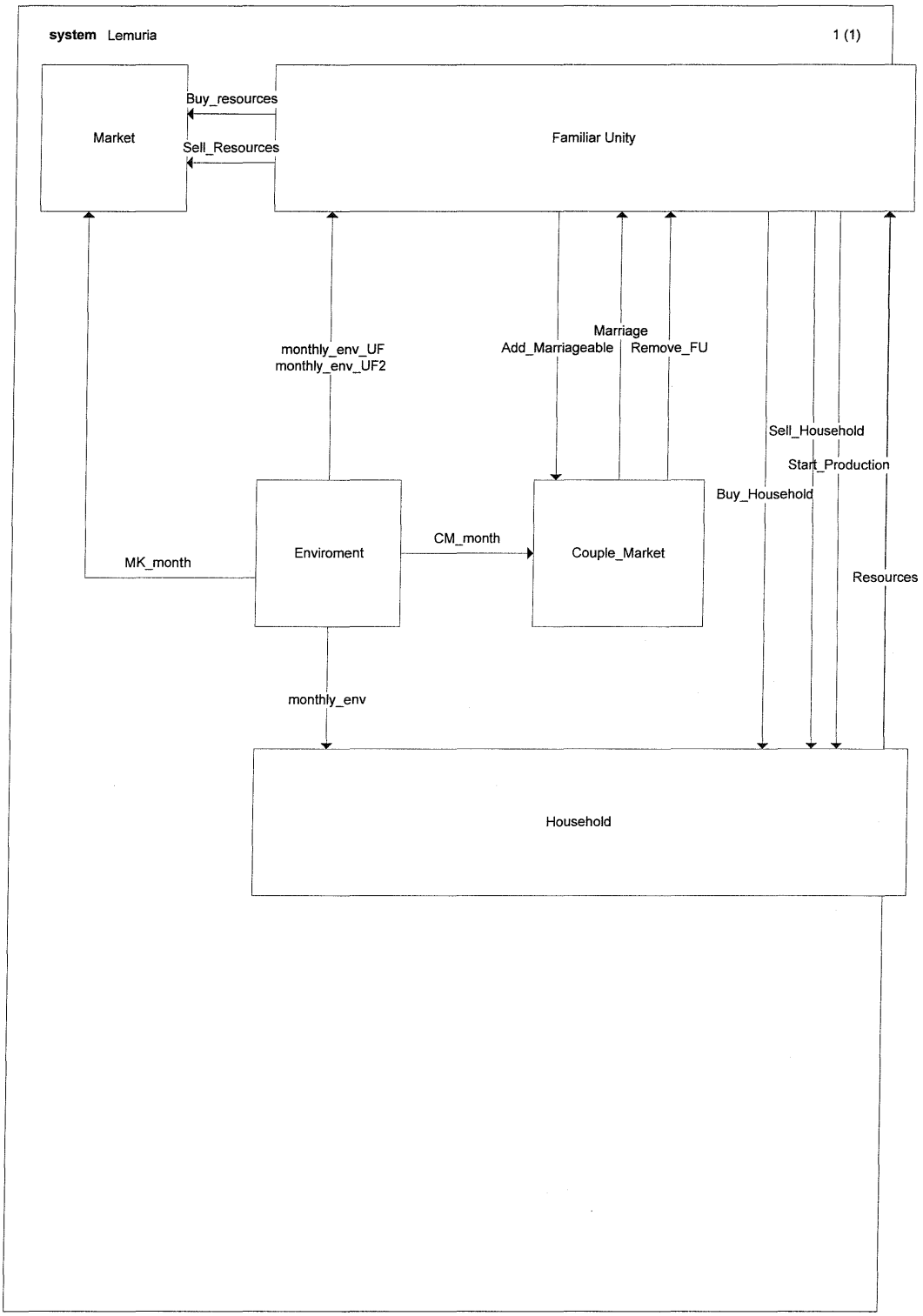
www.itu.int/ITU-T/studygroups/com10/languages/Z.100_1199.pdf

5.2. Disseny SDL

En aquest punt es podran trobar els diversos diagrames SDL que componen el simulador. Donat que de forma natural s'havia dividit la societat medieval en 5 blocs i que l'especificació SDL permet la definició de bloc independents es va crear un bloc SDL per cadascun dels blocs naturals que s'havia especificat quan es realitzava l'estudi de la ciutat medieval.

[18] El SDL-forum és una organització sense ànim de lucre que té per objectiu de promoure l'ús del llenguatge SDL. A la seva web hi ha molta informació referent al llenguatge així com es poden trobar notícies i esdeveniments relacionats amb l'SDL.

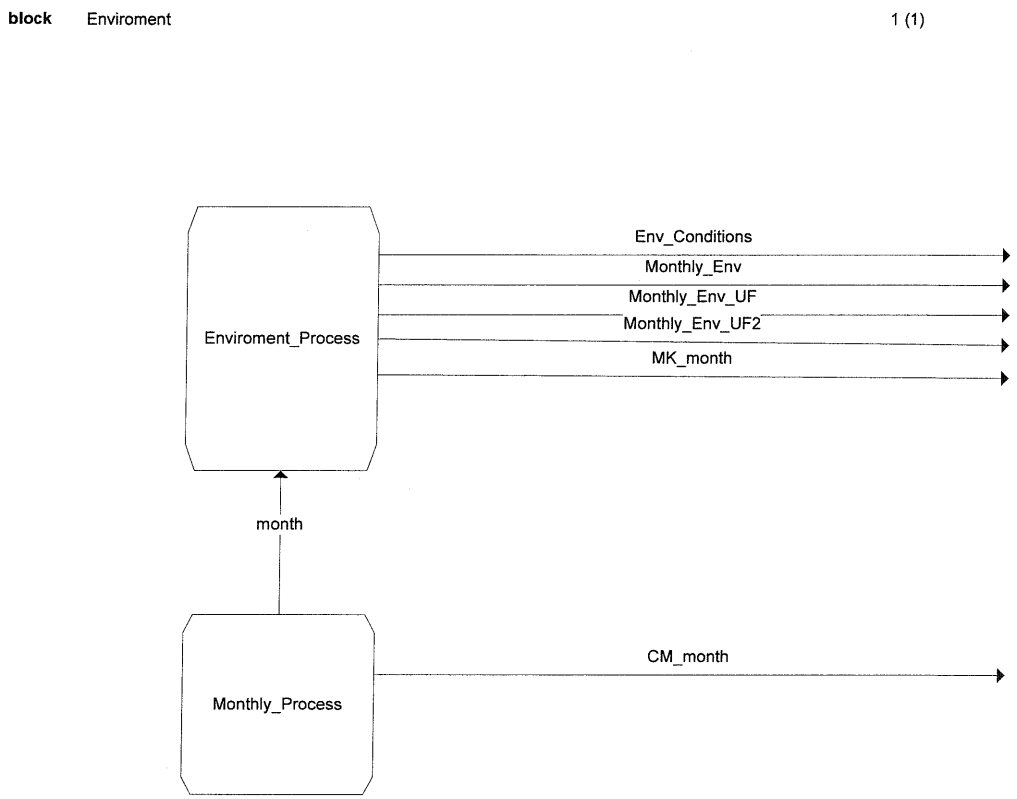
5.2.1. System


system Lemuria
1 (1)
Market
Buy_resources
Sell_Resources
Familiar Unity
monthly_env_UF
monthly_env_UF2
Marriage
Add_Marriageable
Remove_FU
Sell_Household
Start_Production
Buy_Household
Resources
MK_month
Enviroment
CM_month
Couple_Market
monthly_env
Household

Aquest bloc especifica el que serà el nostre sistema de forma global. Es pot veure com dins seu s'hi troben 5 blocs, un per cadascun dels blocs identificats.

També es poden apreciar les senyals que els uneixen els uns amb els altres. Aquestes seran explicades en detall en cadascun dels blocs origen i destí.

5.2.2. Bloc de Clima

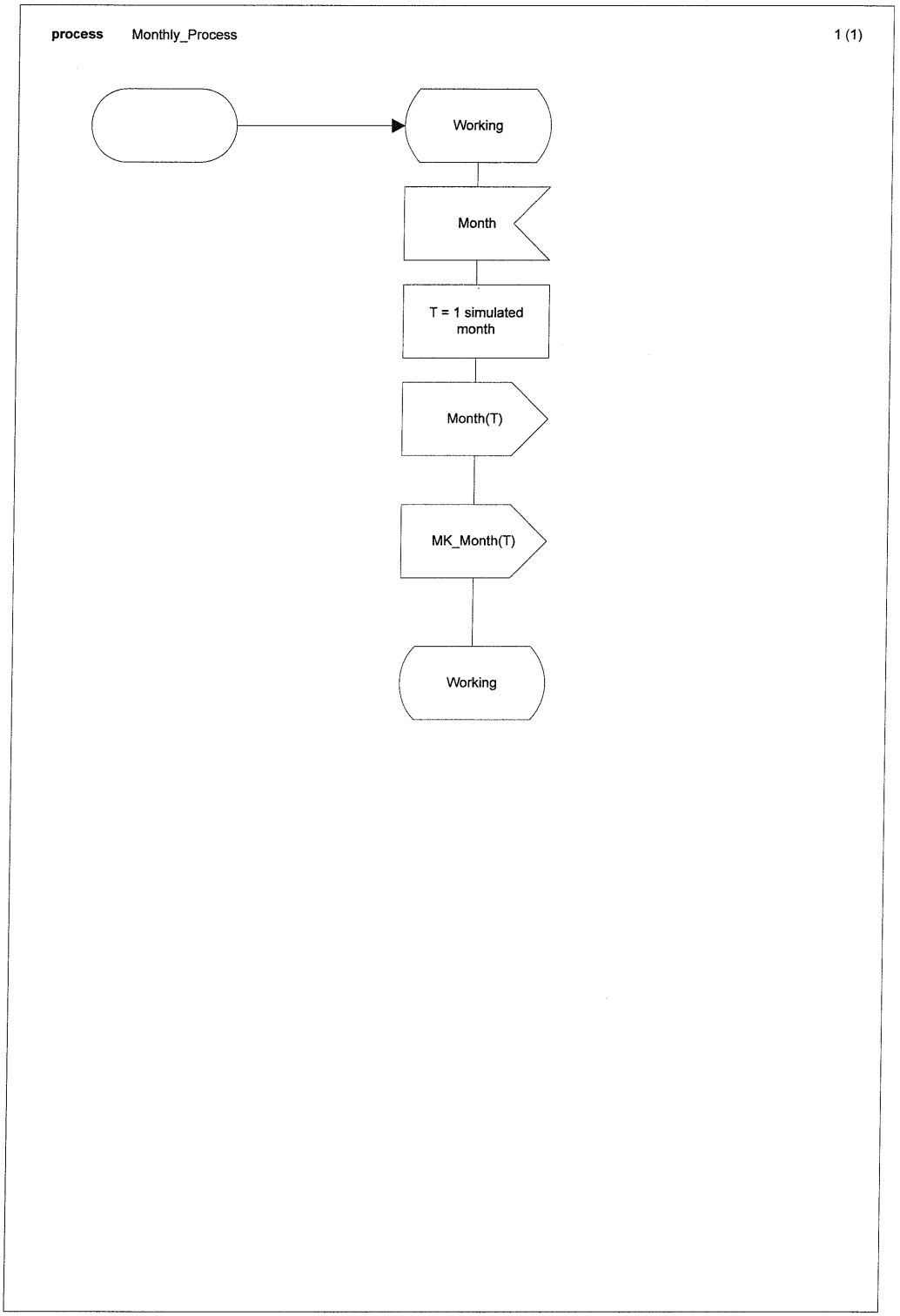


Com es pot veure en el diagrama SDL del bloc de Clima, aquest està format per dos processos: El procés Enviroment i el procés Monthly. El procés Monthly és el procés encarregat de generar una senyal de forma periòdica que marcarà l'evolució del temps en mesos. Cada cop que es generi aquesta senyal, significarà que haurà passat un mes de simulació.

D'aquest procés en surten dos canals. Ambdós transporten la mateixa senyal en el mateix moment i la única diferència que hi ha entre ells és que la que surt pel canal month anirà al procés d'enviroment i la que surt pel canal CM_month anirà destinada cap a fora del bloc, en concret, aquesta senyal anirà al bloc de relacions (Couple market).

A continuació descriurem el comportament i estructura de cadascun dels dos processos .

Procés Monthly

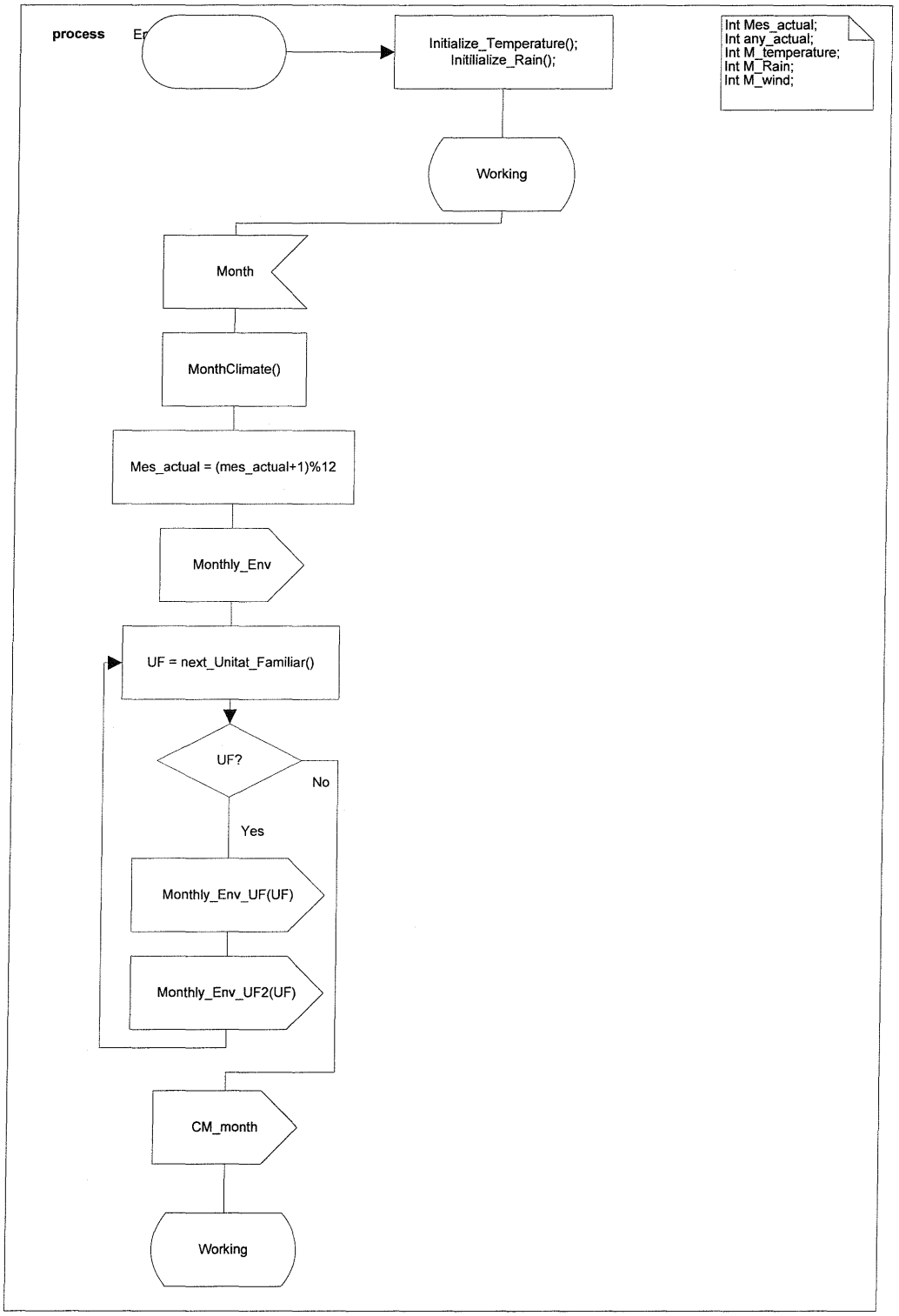


Aquest procés té la funció, com ja s'ha comentat, de generar una senyal mensual. És el més senzill de tots els processos que podrem trobar i va ser el primer de tots a ser implementat.

Com es pot comprovar als diagrames, aquest procés parteix d'un estat inicial 0 i immediatament envia el senyal month, que serà rebut pels blocs Enviroment i per ell mateix. A continuació es situa a l'estat de WORKING_M del que ja no en sortirà mai.

Un cop està en aquest estat i rep el senyal Month (el primer cop rebrà el senyal que ell mateix ha generat mentre s'inicialitzava) generarà un altre senyal Month i un CM_Month amb un lag de 30 dies (són 30 unitats de temps, donat que cada unitat de temps es correspon amb un dia, són 30 dies). A continuació es mantindrà a l'estat WORKING_M.

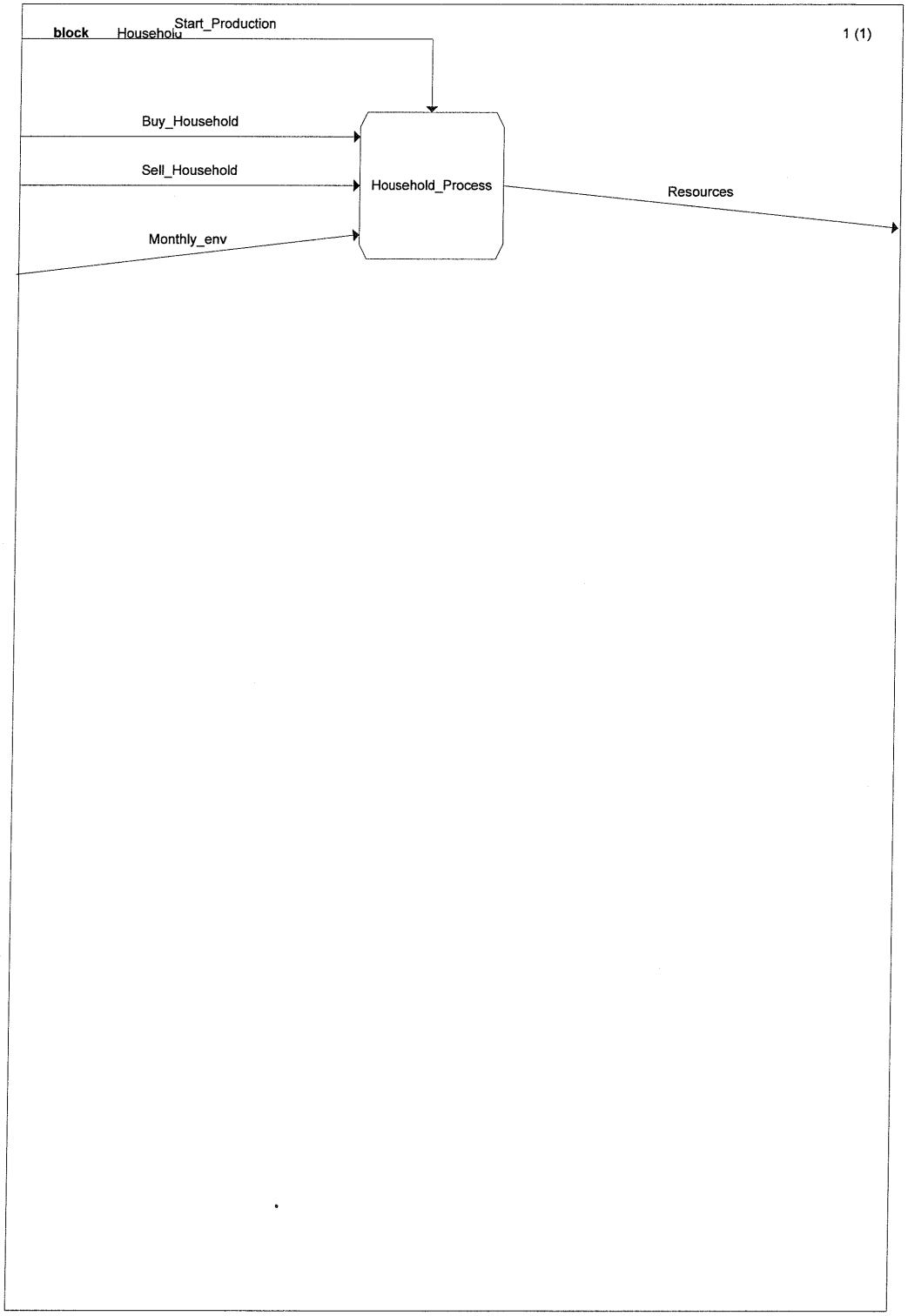
Procés d'Enviroment



Aquest procés és el procés encarregat de generar el clima que afectarà al simulador en el mes següent de simulació. El càlcul del clima es basa, de forma molt simplificada, en calcular a partir de les mesures mitges de pluja i temperatura que regiran durant l'any actual, la temperatura i pluja que afectarà durant el mes següent. D'aquesta manera, podem tenir anys freds , calorosos, plujosos,... Cada mes, també es calcula la quantitat de vent que es donarà. Tota aquesta informació serà enviada a cadascun dels diferents households, cada família i al mercat per a que actuïn en conseqüència.

De la mateixa manera que cada mes es calcula el clima que regirà, cada gener, abans de realitzar el càlcul per al mes actual, es fa el càlcul per a l'any sencer i es calculen, per tant, les mitjanes anuals de pluja i temperatura.

5.2.3. Bloc de Household

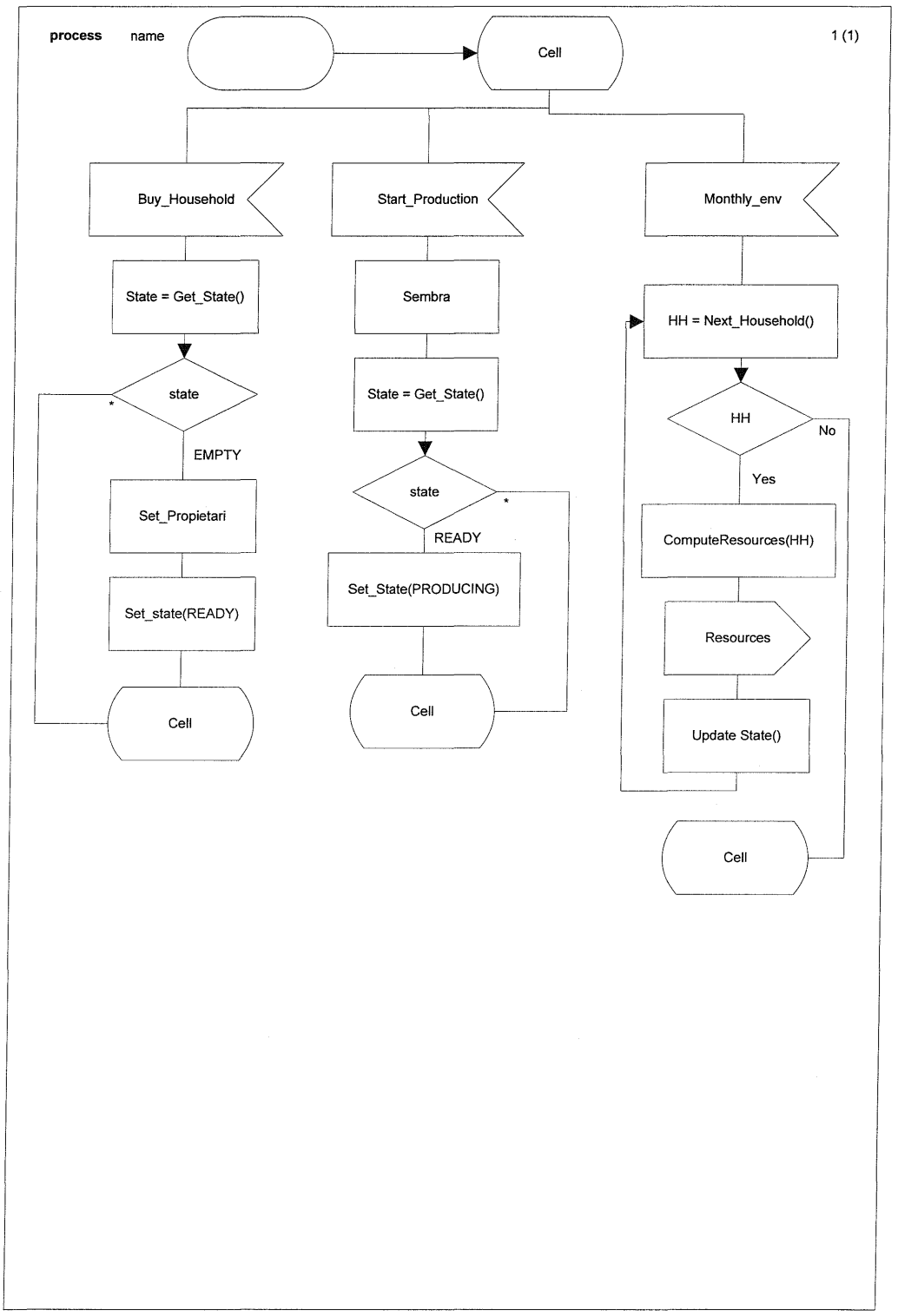


El bloc de Household és el bloc que simula el comportament de les diverses parcel·les que conformen el simulador. És, per tant una de les parts més importants del simulador.

Tot i la seva importància, el seu comportament es centra en un únic procés que controlarà tots els aspectes de totes les parcel·les. D'aquest procés en surten totes les senyals que surten del bloc i li arriben i es processen totes les que rep el bloc.

La motivació de que hi hagi un únic procés és la de intentar simplificar al màxim la complexitat d'aquells punts que estaran més repetits. Pensem que aquest procés es repetirà per cadascuna de les instàncies de household que tinguem, que poden arribar a ser potencialment molt elevades.

Procés Household



Tal com s'explica a l'apartat de desenvolupament del projecte, aquest bloc va patir un problema important a causa de que la prioritat de les senyals no estava correctament implementada. Tant és així, que es va reduir tota la seva complexitat d'estats a un de sol però que tingués un funcionament correcte. A més a més, hi havia també la motivació de simplificar al màxim tots els aspectes complexos del procés, i limitar els estats a un de sol és un dels enfocaments possibles.

Tot i ser un únic procés, existeixen tres comportaments dins seu lligats a una sèrie de senyals cadascun que actuen de forma quasi independent els uns dels altres.

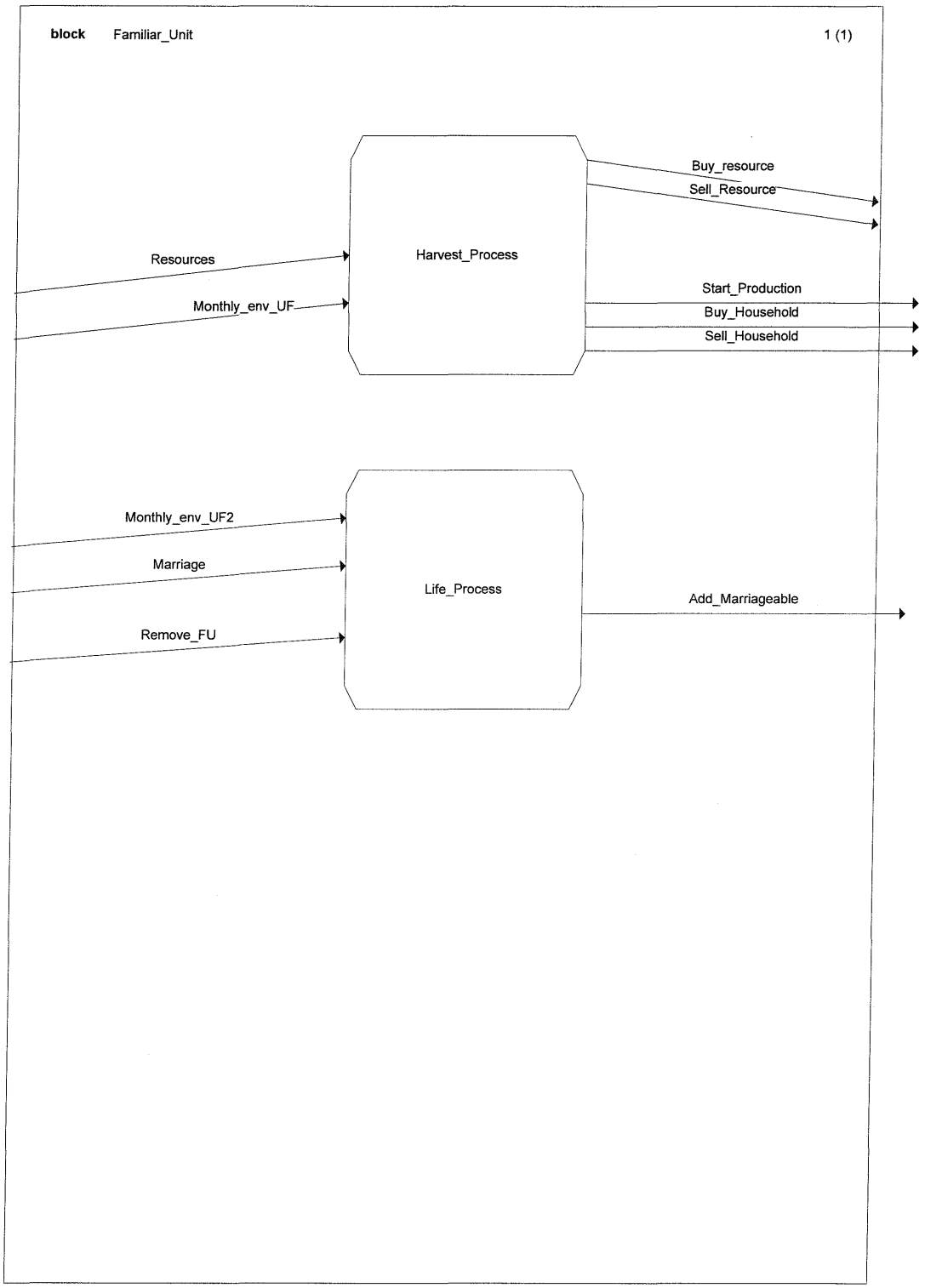
Per una banda tenim la recepció de la senyal amb el clima mensual. Quan es rep aquesta informació, és processada i es realitza el càlcul de com afecta el clima als cultius que s'estiguin realitzant[19].

Per l'altra banda, disposem de les senyals de control per part de la unitat familiar. Comprar (BUY_HOUSEHOLD), Vendre (SELL_HOUSEHOLD) i Sembrar (START_PRODUCTION) parcel·les faran que canviï l'estat intern del bloc i que, per tant, actuï d'una forma diferent i generi unes senyals diferents.

L'últim punt és el punt de la generació de productes. Quan es dona el mes de recollida, el household genera una sèrie de recursos en funció de la producció que se li ha assignat per part de la família i l'efecte que el clima ha tingut sobre la collita que seran enviats a la família propietària mitjançant la senyal de recursos (RESOURCES).

[19] La relació entre clima i conreu ens vindrà donada per la paramterització de la ciutat introduïda.

5.2.4. Bloc de Unitat Familiar

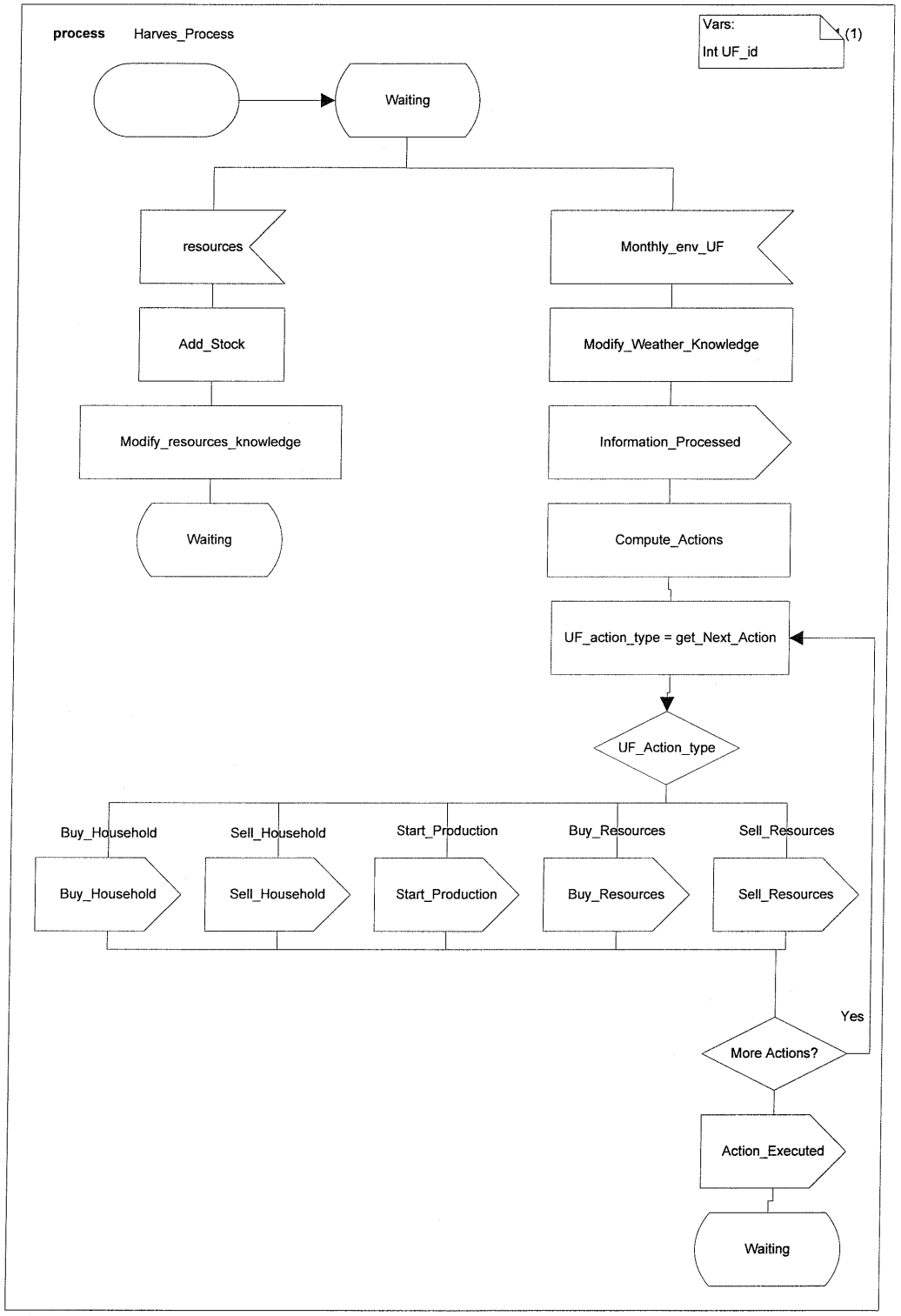


El bloc d'unitat Familiar és el bloc que controla el comportament i la evolució de les famílies que són simulades. És, per tant, evident, que aquest bloc serà el que tindrà més complexitat de tots i on s'allotjarà la intel·ligència artificial i la vida de les persones.

Està format per dos processos: per una banda hi ha el procés de harvest, que s'encarrega de recollir informació del simulador i de calcular quines són les accions que ha de prendre la família de forma anual. A l'altra banda tenim el procés de Vida (Life), que s'encarrega de controlar els naixements, morts, evolucions, alimentació... de les famílies.

A continuació es descriuran els dos processos amb detall.

Procés de Harvest



Aquest procés, com ja s'ha comentat, rep informació de diversos blocs i computa les accions que la família haurà de prendre de forma anual. En concret, aquest bloc rep informació del clima del bloc Enviroment, rep també dels recursos generats pels Households (parcel·les). Amb aquesta informació més la informació que posseeix de la unitat familiar, cada any (en concret, al més de setembre[20]) es genera una llista d'accions que poden ser dels següents tipus:

- Comprar una parcel·la (BUY_HOUSEHOLD)
- Vendre una parcel·la (SELL_HOUSEHOLD)
- Començar a produir algun producte (START_PRODUCTION)
- Comprar algun recurs (BUY_RESOURCES)
- Vendre algun recurs (SELL_RESOURCES)

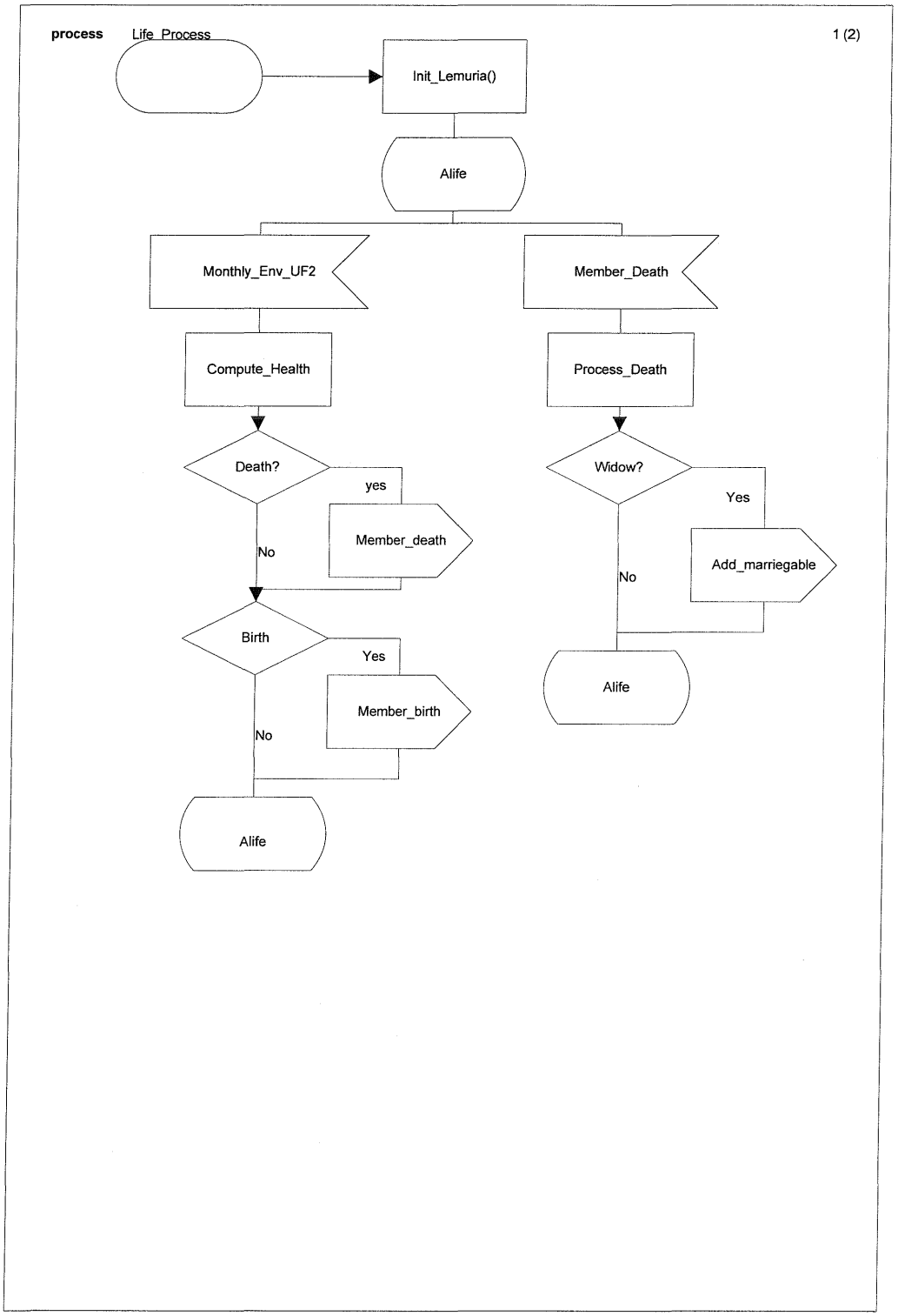
Els tres primers fan referència a tot el tema d'accions que es poden fer amb les parcel·les i, per tant, seran enviats al bloc de Household. Els dos últims, per contra, fan referència a les accions de vendre i comprar recursos al mercat i, per tant, seran enviats al bloc de mercat (veure més endavant).

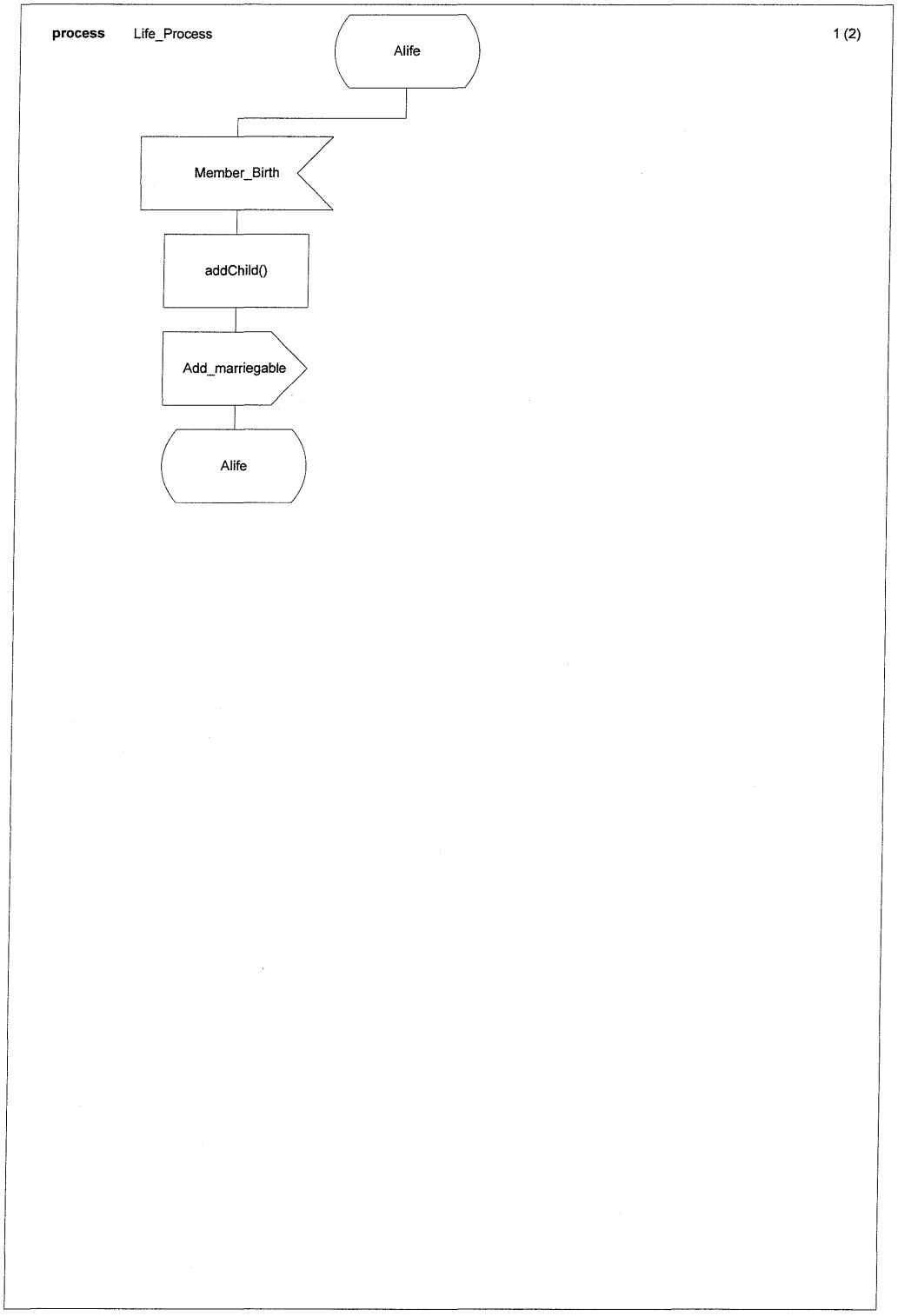
En els diagrames destaca sobretot el bucle final on s'envien les senyals una a una i amb dies de diferència per a evitar possibles conflictes (si una família, per exemple, vol comprar una parcel·la i plantar-hi tomàquets, és important que primer la compri i després hi planti els tomàquets, sinó, la parcel·la no reaccionarà).

D'aquest diagrama també en destaca la crida a la funció de la IA. Aquesta crida compute_Actions és on hi ha i on hi haurà codificada la IA que regirà el programa. Aquesta genera una llista d'accions que seran consultades una a una per el bucle posterior, convertides en senyals i enviades als bloc corresponents per a que actuïn en conseqüència.

[20] Segons la documentació històrica, el mes de setembre era el mes de més activitat a l'edat mitjana. En aquest mes, després de les collites del blat i les verdures (que es feien al agost) era quan es procuïa la major quantitat de transaccions econòmiques i hi havia més activitat fora dels camps.

Procés de Life





El procés de Health és el bloc que, com ja s'ha comentat, controla tot l'aspecte de la salut, naixements i morts que es produeixen dins de la unitat familiar. Destaca també que és aquest

bloc el que, en inicialitzar-se, carrega totes les dades del simulador. Això es fa aquí per comoditat ja que les inicialitzacions del simulador podrien trobar-se a qualsevol punt del mateix.

Aquest bloc pot rebre senyals tant de si mateix (MEMBER_BIRTH, MEMBER_DEATH) com de blocs externs (rep el clima del bloc Enviroment i la notificació quan es produeix una parella nova del bloc de relacions).

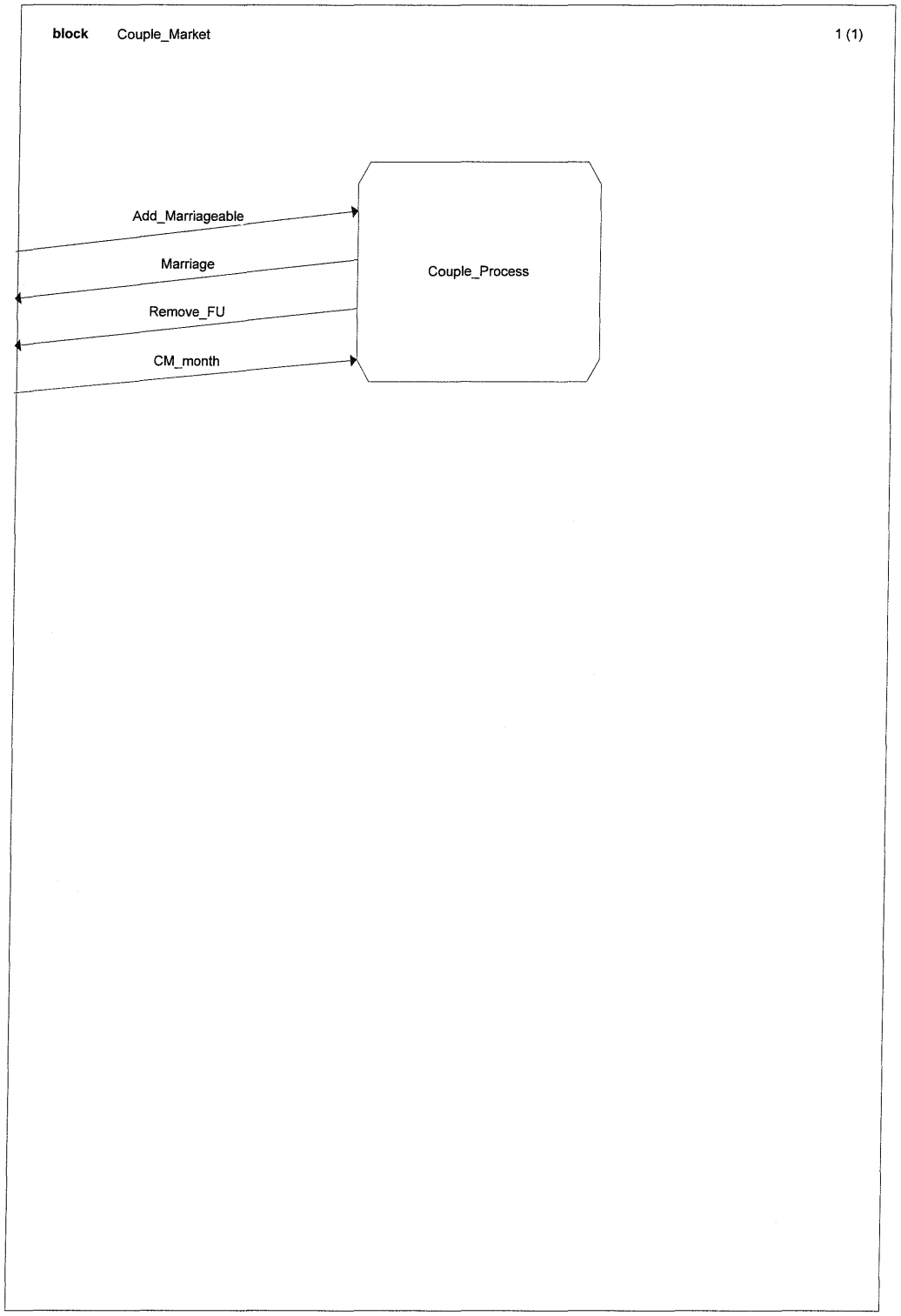
En general és un bloc que tindrà un comportament independent per a cadascuna de les famílies. S'articula, sobretot, a partir del més de gener, quan es fa el càlcul de la nutrició de la família i es calcula també la possibilitat de que hi hagi hagut alguna mort o algun naixement. Si es dona alguna de les circumstàncies, aleshores es procedirà a enviar la senyal MEMBER_BIRTH o MEMBER_DEATH i seran processades pel propi bloc més endavant.

A més a més, al mes de gener s'incrementa l'edat de tots els membres de la família.

Quan neix un nou membre a la família, aquest serà notificat al bloc de relacions (couple_market) per a que li busqui alguna parella adequada en el moment adequat (parlarem d'aquest tema més endavant, en el bloc de relacions). Quan es troba la parella, el procés de Health rep el senyal MARRIAGE que li indica a la unitat familiar que aquest membre l'abandona per a formar una família nova.

La funció del càlcul de la salut és la compute_health i retornarà dos valors, l'un per a indicar si hi ha hagut alguna mort, i l'altre per a indicar si hi ha hagut algun naixement. Si una família es queda sense progenitors, aleshores un dels no progenitors que siguin major d'edat passarà a ser el progenitor de la família. Per contra, si no hi ha cap membre major d'edat, la família mor.

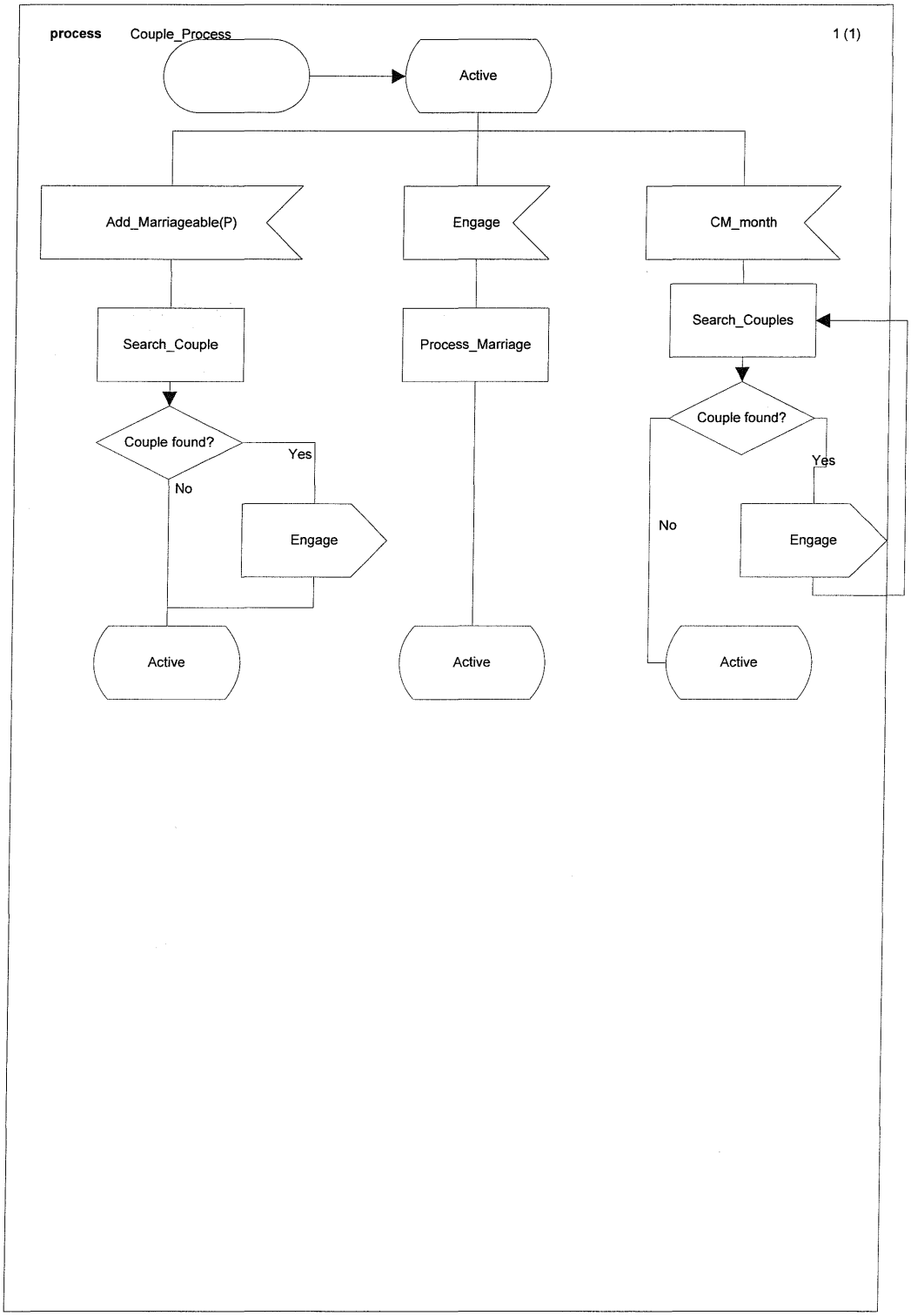
5.2.5. Bloc de Relacions



El bloc de relacions és el bloc que s'encarrega del tema de l'establiment de les parelles per a formar noves unitats familiars. Anomenat Couple_Market perquè la seva funció és la de "vendre" les persones a la parella més adient.

Dins seu trobarem un únic procés que rep senyals del bloc enviroment i de la unitat familiar i envia informació a la unitat familiar per a notificar la creació de les noves parelles.

Procés Couple Market

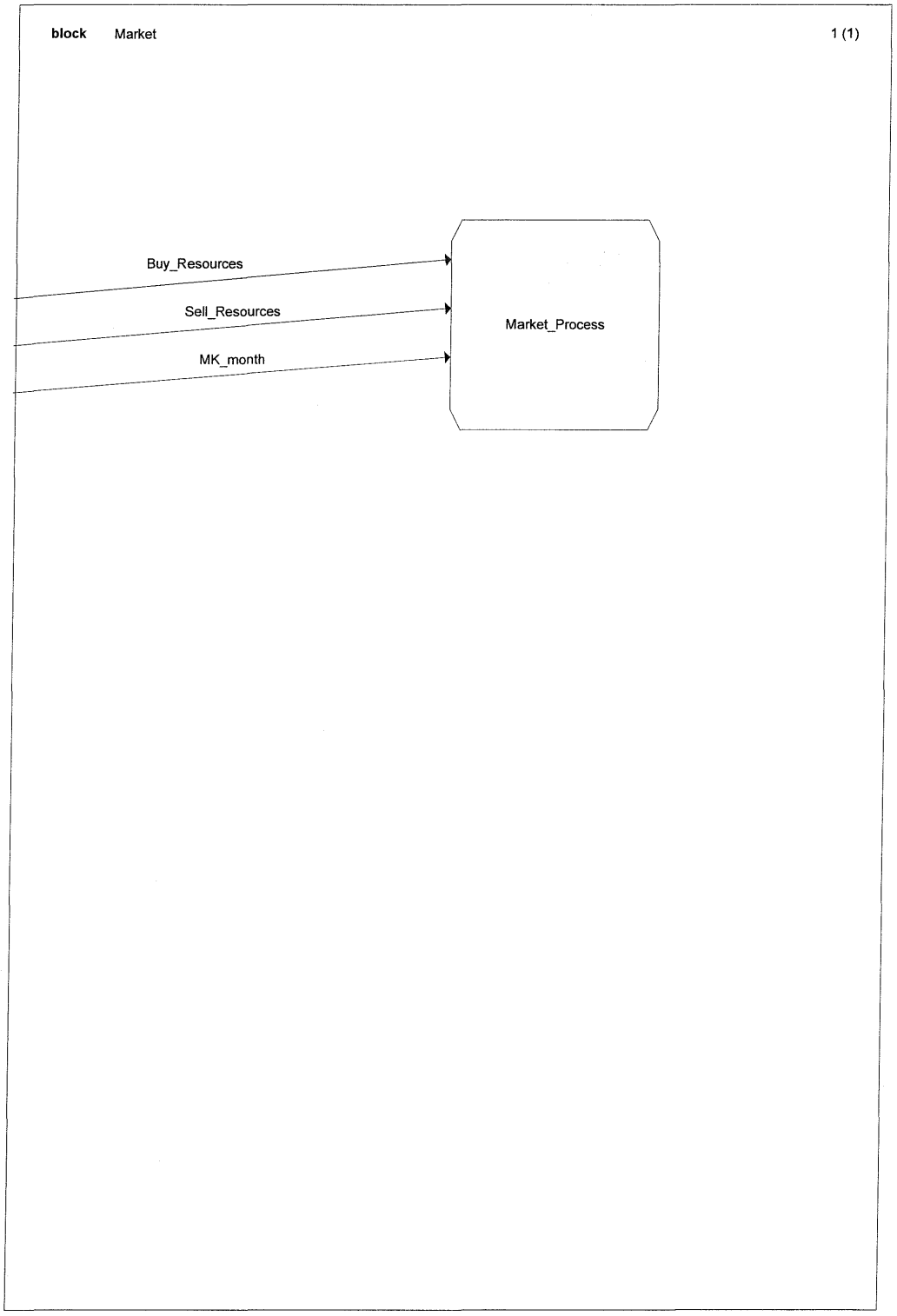


Aquest procés és el nucli del bloc. Li arriben totes les senyals del bloc i d' ell surten totes. En concret, rep informació mensual del bloc enviroment. Amb aquesta informació, el que farà serà periòdicament (al gener de cada any) analitzar el conjunt de solters del simulador i crear les parelles adequades.

Per tal de tenir un control dels solters del simulador, el procés rep la senyal Add_marriegeable, que surt del bloc de unitat familiar. Aquesta senyal porta incorporada informació del membre solter que volem "posar en venda".

Un cop s'ha creat una parella, s'envia la senyal de marriage al bloc de unitat familiar amb la informació dels membres que formaran la nova família per tal de que la unitat familiar rebi aquesta informació i la processi adequadament. A més, els dos membres de la nova família seran esborrats de la llista de solters.

5.2.6. Bloc de Mercat

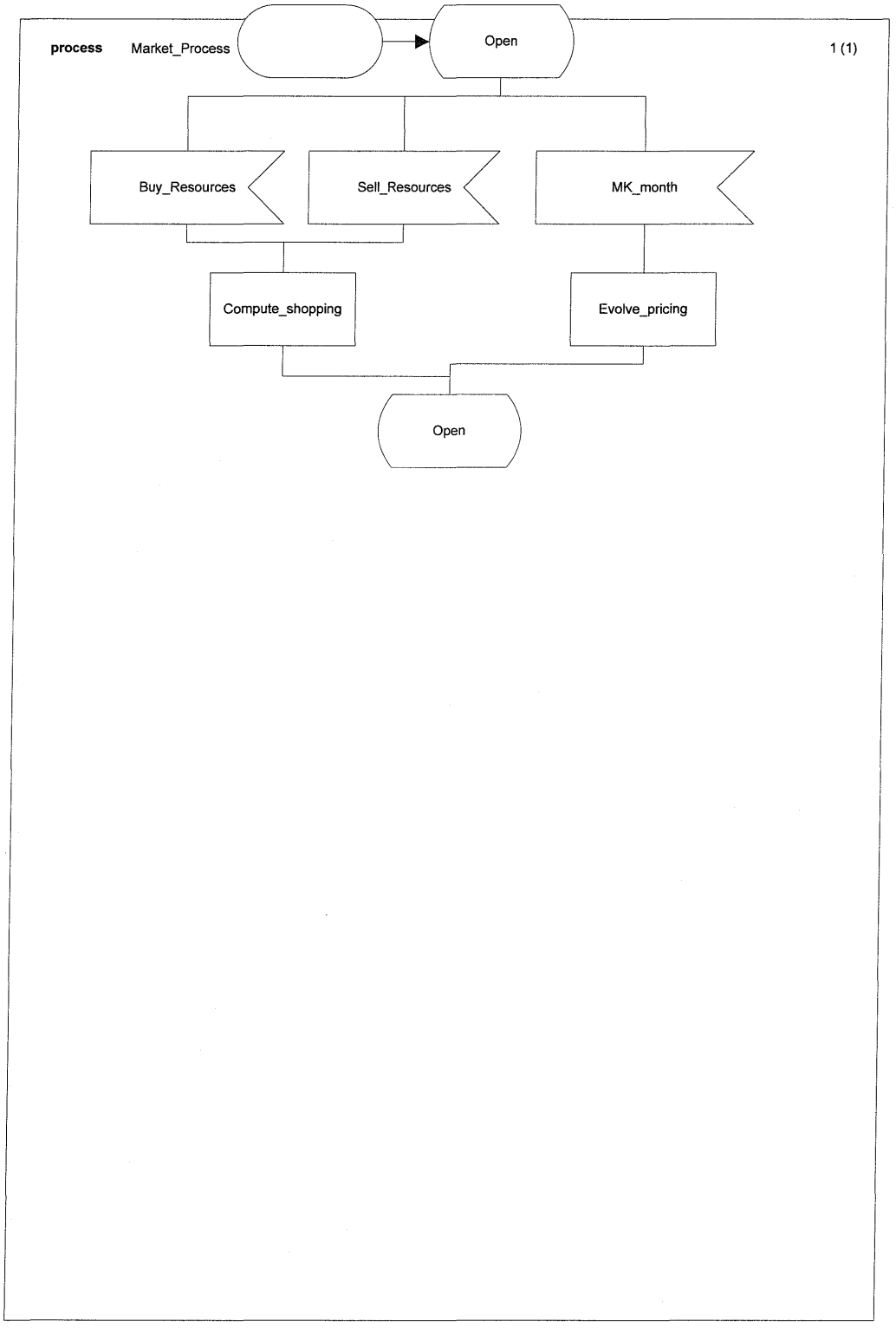


El bloc de mercat és l'encarregat de la compra i venda dels productes que produeixen les famílies. Rep informació del bloc de clima per tal de realitzar la tasca d'ajust de preus de forma periòdica (al mes de juny de cada any[21]). I de la unitat familiar per a la compra i venda que vulguin realitzar dels diversos productes.

Conté un únic procés que serà el que controlarà tots els aspectes del bloc.

[21] El mes escollit ha estat el juny perquè és el mes anterior a la recollida de tots els productes agrícoles. La tasca, però, es podria realitzar qualsevol altre mes o, amb algunes modificacions del codi de les funcions, de forma mensual.

Procés de Mercat

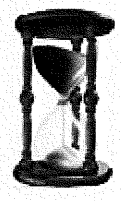


Aquest procés rep tota la informació del bloc i envia totes les senyals d'aquest.

Li arriba la senyal mensual del bloc de clima i de forma periòdica, al juny, fa el recàlcul dels preus per tal d'ajustar-los a la oferta/demanda.

A més a més, rep informació dels productes que volen vendre les famílies a través de les senyals SELL_RESOURCES i BUY_RESOURCES. Aquestes senyals porten informació dels productes que es volen vendre i/o comprar i el mercat s'encarregarà d'incorporar-ne les dades per al càlcul de la oferta i demanda de l'any següent.

Cal destacar aquí que el bloc de mercat no envia cap senyal a les famílies per a notificar-los les vendes del seus productes. Això es deu al fet de que les famílies poden consultar la informació de la ràtio de venda d'un producte determinat en qualsevol moment i calculen la quantitat venuda i/o comprada dels productes de forma automàtica.



En un futur, s'afegirà la funcionalitat de que el bloc de mercat comuniqui al bloc de unitat familiar la quantitat exacte dels seus productes que s'han venut / comprat. Aquesta nova funcionalitat serà útil en el moment en que disposem de productes de diferents qualitats que els diferenciin segons la família que els produeix.

5.3. XML generat

Un cop es tenien els diagrames elaborats, per tal de poder-los introduir en el SDLPS s'havia de crear un fitxer XML que contingués la informació dels mateixos. Aquest XML té una correspondència 1:1 amb els diagrames i es pot consultar a l'annex Model Lemuria.xml (pag 131).

6. Hipòtesis Simplificadores

Un dels punts més important en la elaboració d'un simulador és la de simplificar la gran complexitat del món real per tal de poder realitzar una eina adequada. És també important tenir en consideració quines són les simplificacions que s'han fet per tal de conèixer les limitacions de la eina i quines poden ser les futures ampliacions de la mateixa. A aquestes simplificacions se les anomena Hipòtesis simplificadores i a continuació es presenta una llista de les que es corresponen amb el projecte:

6.1. Genèriques

- Cada mes té 30 dies. Cada any té 12 mesos. Per tant, cada any té 360 dies.
- Els elements que es contemplen del clima són la pluja, el vent i la temperatura.
- No es contemplen desastres naturals.
- No es contemplen canvis bruscos del clima que durin menys d'un mes. Suposarem que la temperatura es manté constant al llarg d'un mes sencer.

6.2. Sobre el comportament de les persones

- Els embarassos duren un any sencer.
- Les persones passen per tres fases al llarg de la seva vida: Infant, Adolescent i Adult. Es considera Infant a les edats entre 1 i 12 anys. Adolescent entre 13 i 18 i adult a partir d'aleshores. No es contempla la vellesa.
- Totes les persones són fèrtils.
- No es contempla el divorci.
- No es contempla la possibilitat d'embarassos fora del matrimoni.
- No es permet casar-se a persones que no siguin adultes.

- Els infants i adolescents no poden tenir fills.
- Les famílies es poden endeutar durant un any. Un cop acabat aquest, hauran de retornar el deute venent els seus recursos al mercat sense cobrar.

6.3. Sobre els elements del simulador

- Per a realitzar les simulacions s'ha optat per a definir 6 recursos bàsics que són els següents:
    - Cereals: Que inclourien tots els cereals cultivables i les farines que se'n deriven
    - Verdures: Inclouen la fruita i verdura que es consumia en la zona estudiada.
    - Carn: Que inclou tot el que seria carn tant de ramat (boví, oví..), aviram i pesca.
    - Roba: Inclou tots els elements necessaris per a la persona: Roba, guants, barrets...
    - Eines: Inclou tots els elements necessaris per al bon funcionament d'una llar. (Terrissa, estris de cuina,...)
    - Objectes de Luxe: Inclou tot allò que no era necessari però que aportaven estatus social i major satisfacció a les famílies. (Joies, espècies, roba cara,...)
- No es considera l'obtenció de menjar per caça o recol·lecció dels boscos (fruits salvatges, bolets,...)
- Tota parcel·la produeix la mateixa quantitat d'un mateix producte. No es te en compte que hi hauria parcel·les de terreny més apropiades per a un cultiu que per a un altre.
- El recàlcul de preus dels productes del mercat es faran de forma periòdica cada any i tenint en compte la producció, compra i venda de l'any anterior.
- El clima afecta a totes les parcel·les per igual.
- Els productes manufacturats no tenen requeriments de productes bàsics per a ser elaborats.

7. Decisions tècniques

7.1. Plataforma i Software

Quan es plantejava la creació del projecte Lemúria, ens vam plantejar quina hauria de ser la seva estructura bàsica i sota què havia de funcionar. Existien dues alternatives diferenciades sobre quina era la filosofia que havia de seguir: Per una banda es tenia la opció de crear-lo utilitzant el software SDLPS i per l'altra, utilitzant programació en C/C++ utilitzant la llibreria CUDA[22].

La primera opció, la opció del SDLPS era una opció que tenia les següents característiques:

- Permetia integrar directament dissenys SDL sense necessitat de passar per una codificació intermèdia
- Funcionava sota Windows XP/Vista independentment del hardware utilitzat.
- S'està desenvolupant amb propòsit acadèmic i de investigació.
- Jo disposava d'experiència treballant amb C/C++ i XML, cosa que facilitava el fet de treballar sota aquestes tecnologies.
- Donava la possibilitat de distribuir l' execució en diversos equips.

La segona opció, la de la llibreria CUDA tenia les següents característiques:

- Segons la documentació consultada a la web oficial, permetia increments del rendiment que podien arribar a ser entre 1000 i 10000 del que s'obtenia funcionant sobre un únic processador.

[22] La llibreria CUDA es un conjunt d'eines desenvolupades per NVIDIA que permeten la creació de software per a que s'executi utilitzant la GPU enlloc de la CPU. El gran avantatge que te és que permet un nivell de pararl·lelisme sorprenent (entre 128 i 1024 processos funcionant concurrentment en els processadors de la targeta gràfica). Per a més informació, consultar http://www.nvidia.com/object/cuda_home.html

- La llibreria i el codi són de lliure distribució i ve integrada amb les targetes de última generació de NVIDIA.
- Només funciona en alguns models del targetes gràfiques de NVIDIA. La llista completa es pot consultar a http://en.wikipedia.org/wiki/CUDA#Supported_GPUs.
- Els programes creats sota CUDA poden funcionar potencialment sota Windows XP/Vista, Linux i Mac OS X.

La tria va ser difícil i va portar algun mal de cap. Per una banda, el simulador era un projecte ambiciós, cosa que feia que utilitzar el SDLPS simplifiqués, tant a nivell d'implementar el disseny com a nivell de tecnologies emprades. Per altra banda, sabia que el projecte requeriria una gran quantitat de processament de dades, cosa que faria que jo no pogués llençar simulacions gaire complexes si no es que utilitzava la potència que em podia proporcionar CUDA.

La resposta va venir de la ma del hardware i la distribució:

La meva idea era fer un projecte que es pugui utilitzar en un futur per a fer estudis sociològics, i això requereix poder distribuir la eina i que es pugui executar en la majoria de màquines. És veritat que l' SDLPS és només per a Windows[23], però el fet de que la llibreria CUDA es pogués executar només sota algunes targetes NVIDIA em va fer veure un primer principi d'inviabilitat en utilitzar la llibreria de NVIDIA.

A tot això es va sumar el fet de que jo disposava en aquell moment de dos ordinadors on podia treballar en el projecte: Un ordinador de sobretaula amb una targeta NVIDIA compatible amb CUDA i un portàtil amb una targeta gràfica ATI (incompatible amb CUDA, per tant). Això volia dir que el desenvolupament del programa l'havia de fer únicament amb el meu equip de sobretaula, cosa que em limitava la possibilitat de treballar fora de casa (per raons alienes al projecte, durant aquesta època havia de viatjar prop de 2:30 h en tren cada dia i pretenia aprofitar aquest temps) i que si tenia algun problema amb ell (que ja n'havia tingut algun feia poc), em quedaria parat en el procés de desenvolupament de la eina.

L'últim element que em va fer decidir va ser el fet de que, tot i que no tindria velocitat quan treballés amb el meu equip personal, l' SDLPS permetia la interconnexió de múltiples equips

[23] Es pot executar sobre Linux amb Mono o recompilant el projecte.

per a la simulació, cosa que feia que, si en un futur es necessitava llençar una simulació a gran escala, el programa estava preparat per a fer-ho de forma eficient utilitzant diversos equips.

7.2. SDL vs. Llenguatge de programació

Tal com s'ha explicat la secció del SDL, aquest disposa d'un últim nivell de diagrames (els que implementen les operacions que van dins del requadres en els diagrames de transicions) que es pot optar per a fer-los en SDL o en algun llenguatge de programació com C, C++, Java...

Les operacions que havia de fer servir eren de gran complexitat, i ja des del començament tenia jo molt clar que no volia utilitzar el SDL, però no obstant, em vaig plantejar les dues opcions.

La opció SDL tenia els següent factors:

- Es podien especificar en els diagrames de disseny SDL i, per tant, es podien integrar dins de l' XML que es creava a partir del disseny.
- No requereix compilació ni tractament extra.
- No estava implementat al SDLPS.
- Operacions vitals com Create o Delete d'entitats no estaven implementades.

La opció de utilitzar algun llenguatge de programació tenia els següents factors:

- El resultat obtingut era més versàtil que no pas si es dissenyava en SDL.
- Si utilitzava algun llenguatge eficient com C o C++, el rendiment seria major.
- La opció d'integrar codi extern no estava implementada al SDLPS.

Veient les dues opcions, era evident que implicaven que algun canvi s'hauria de fer al SDLPS si es volia completar el projecte. No obstant, la opció de integrar codi extern es presentava més senzilla donat que el propi SDLPS ja disposava d'opcions per a compilar els fitxers C que ell mateix generava.

Per altra banda, jo no disposava d'experiència en el desenvolupament de les operacions sota el paradigma SDL i, en canvi, tenia l'experiència de tota la carrera en desenvolupament de funcions i operacions en llenguatges com C, C++ o Java.

A aquest fet es suma el que algunes operacions que jo necessitava (Create i Delete, principalment) no estaven implementades al SDLPS i, per tant, si les volia fer servir les havia d'implementar o havia de crear-les de forma particular al codi extern. Si analitzem la gran complexitat d'ambdues operacions, veurem com si les hagués hagut d'implementar al SDLPS hauria requerit bastant temps que hauria hagut de sacrificar del desenvolupament de la resta del projecte.

Com a últim punt, el rendiment que podia obtenir i el control que tindria a nivell de memòria i, d'optimització del codi si treballava sota C o c++ era molt superior al que obtindria si treballava sota SDL.

Tot i que era la mateixa idea que jo creia de bon principi que seria la bona, utilitzar un llenguatge de programació va resultar ser la opció que, a priori, i després de realitzar les reflexions anteriors, es presentava com a millor.

7.3. Llenguatge de programació

Amb el coneixement de que havia d'utilitzar algun llenguatge de programació, arribava el moment de triar quin seria el llenguatge que havia d'utilitzar.

Les opcions que em vaig plantejar en principi van ser el C i el C++. Vaig descartar la resta de llenguatges donat que la opinió majoritària és que tenen falta d'eficiència o de gestió de la memòria (volia tenir control sobre què hi havia exactament a la memòria en cada moment, què entrava i què sortia i què es quedava).

El C++ ens dona la orientació a classes mentre que jo crec que llenguatge C ens dona un rendiment una mica superior.

La pregunta que jo em vaig plantejar era si realment necessitava les classes.

Com es veurà més endavant no és un requeriment, donat que el SDL és un llenguatge orientat a classes i, per tant, la concepció de classes ja ve determinada pel propi disseny SDL. Com es pot veure a la secció del SDL, existeixen una sèrie d'objectes, que són els Blocs i els processos que, en el fons, vindrien a representar les classes que es podrien utilitzar en C++. De fet, si ens hi fixem, l' SDL ja especifica que la única manera de accedir al codi extern és a través de les operacions de crida a funcions, que estan a l'últim nivell i que MAI es comuniquen ni entre elles ni amb la resta de blocs. Vist així, sembla evident que no és

necessari disposar de classes al nivell d'operacions i que tot el codi extern es limitarà a funcions independents que actuaran de forma independent (a nivell de bloc) i només compartint les variables del bloc o procés al qual pertanyin.

Sota aquest context, no obstant, i mantenint encara el meu escepticisme sobre el fet de NO utilitzar classes, vaig intentar de fer proves d'integrar codi en C i en C++ al propi simulador. El resultat va ser que la integració del codi C es va fer sense problemes i, en canvi, integrar el codi C++ va resultar impossible.

Amb tots aquests elements a la mà, la opció triada per implementar el codi dels blocs task (crides a funcions) va ser la del C.

8. Fases del Projecte

El projecte està dividit en tres fases principals més una prèvia de documentació. Aquestes fases marcaran el ritme, les dates i els passos en la elaboració del projecte.

A continuació es pot consultar el plantejament inicial de les diverses fases. En cas de voler més informació, referir-se a l'apartat d'Evolució del projecte.

Nota: S'ha decidit explicar les fases del projecte i mostrar els seus calendaris abans de la implementació i les dades donat que el projecte es va desenvolupar d'aquesta manera, és a dir, es van crear primer les fases i es van planificar i després es van implementar, i perquè és vital conèixer com està estructurat el projecte per a poder entendre correctament com es van dur a terme la implementació i les simulacions.

8.1. Fase prèvia: Documentació

8.1.1. Planificació

A partir del dia 12 de febrer de l'any 2008 fins al l'1 de Maig de 2008 vaig estar realitzant les últimes assignatures de la carrera pel matí i a més vaig estar treballant per les tardes. Això evitava que pogués dedicar-me en profunditat al projecte. Com a conseqüència d'això, em vaig centrar en la labor de recopilar informació que em seria necessària de cara a la creació de la eina. Aquesta prèvia es va centrar, per tant en el següent.

- Estudi de la societat medieval.
- Estudi d'economia bàsica d'oferta, demanda i evolució de preus.
- Estudi Bàsic de Nutrició i Alimentació (Aquest punt és important donat que a l'edat mitja l'alimentació era un punt clau per a determinar qui vivia i qui moria).
- Obtenció de dades climàtiques i socials de l'edat mitjana (El clima a l'edat mitja era sensiblement diferent de l'actual).
- Ampliació de l'estudi del paradigma SDL.

- Repàs d'algorítmica i Intel·ligència artificial.

8.1.2. Desenvolupament de la fase

Aquesta fase va començar oficialment el dia 20 de febrer. Donat que en aquest mateix moment jo estava finalitzant les últimes assignatures de la carrera i estava treballant a mitja jornada, es va decidir que aquesta fase seria molt suau a nivell de hores.

La seva funció era la de recopilar informació sobre tots els temes que tractaria al llarg de les següents fases.

Nota: En aquest punt no es donaran les dades recopilades, si es volen consultar, per favor referir-se als annexos corresponents.

Les primeres hores es van dedicar a la consulta d'informació d'àmbit generalista per Internet del tema de la ciutat medieval, estructura social...

També a través de la xarxa vaig aconseguir recopilar informació de psicologia i conducta humana i sobre la organització física de les ciutats.

Un cop vaig tenir una idea bàsica del funcionament medieval, vaig passar a la fase d'aconseguir informació d'altres fonts que no fossin a la xarxa.

Em vaig posar en contacte amb una professora d'història i em va proporcionar documentació variada i més avançada sobre la estructura social econòmica i política de les ciutats medievals així com informació de la seva dieta, costums, gustos i aficions.

La setmana següent em vaig reunir amb la Dra. Rello de l'hospital Clínic per a parlar de la nutrició de l'ésser humà, l'alimentació i el consum que deurien tenir les persones durant la època medieval. De la seva reunió en vaig treure un conjunt d'apunts i de taules que en el futur em servirien per a simular l'alimentació, salut i consum de les persones de la edat mitja.

La tercera fase de documentació es va centrar en aconseguir informació del cultiu dels diversos productes. Per a fer-ho vaig consultar al meu àmbit familiar, que tenen coneixements de conreu i cultiu de diversos productes agrícoles.

La última part de la documentació va ser la recopilació de dades climatològiques i de producció dels camps. Volia documentar-me a l'arxiu històric de Girona donat que volia realitzar els testos sota la aquesta ciutat, però la inconveniència d'horaris i vacances va fer que això no es pogués produir.

Per a suplir aquesta manca d'informació vaig consultar les dades meteorològiques de l' institut català del temps.

Tota la informació la vaig recopilar i es pot consultar tant als annexos corresponents com a la secció de bibliografia.

8.2. Primera fase: Disseny

8.2.1. Planificació

La primera fase del projecte comprèn el període entre Juny de 2008 i començaments de Setembre de 2008 i es va centrar en la creació del disseny dels diagrames SDL del simulador.

Es va completar amb èxit en el termini planificat tot i que, com era d'esperar, la implementació de la eina va fer sortir a la llum alguns punts conflictius que no havien estat considerats amb anterioritat i que es van incorporar al disseny durant la implementació.

8.2.2. Desenvolupament de la fase

El disseny del projecte es va dur a terme sense excessives complicacions i sota les dates especificades. El primer pas va ser la instal·lació de les eines que necessitava, en aquest cas, el MS Visio i el plug-in Sandrila. Un cop les eines van estar instal·lades i funcionant correctament, es va procedir al disseny del simulador.

La configuració dels diversos blocs es va forjar ja de bon principi, tot i que en un primer moment hi havia 6 blocs enlloc de 5.

Es va pensar en incloure un bloc que controlés els membres de forma independent a la resta de blocs. No obstant, sota recomanació del tutor del projecte, es va decidir prescindir d'aquest

bloc i integrar-ne el comportament dins del bloc de la unitat familiar per tal de no tenir un simulador extremadament complexa.

Un cop la primera idea dels blocs va estar forjada, es va procedir a la elaboració dels primers diagrames d'estats i de transicions. Un a un van ser refinats, corregits o eliminats fins a arribar a obtenir una solució molt semblant a la que al final es podrà veure.

En resum, va ser una fase de reflexió, concentració i concepció d'idees. No es pot subdividir en subfases ni la seva explicació pot ser gaire precisa donat la pròpia natura caòtica de la creació d'un concepte des de zero.

8.3. Segona Fase: Implementació

8.3.1. Planificació

La segona fase comprèn entre els dies 18 de Setembre de 2008 fins al dia 28 de Novembre de 2008. Consisteix en la implementació i primer testeig de la eina dissenyada en la fase anterior.

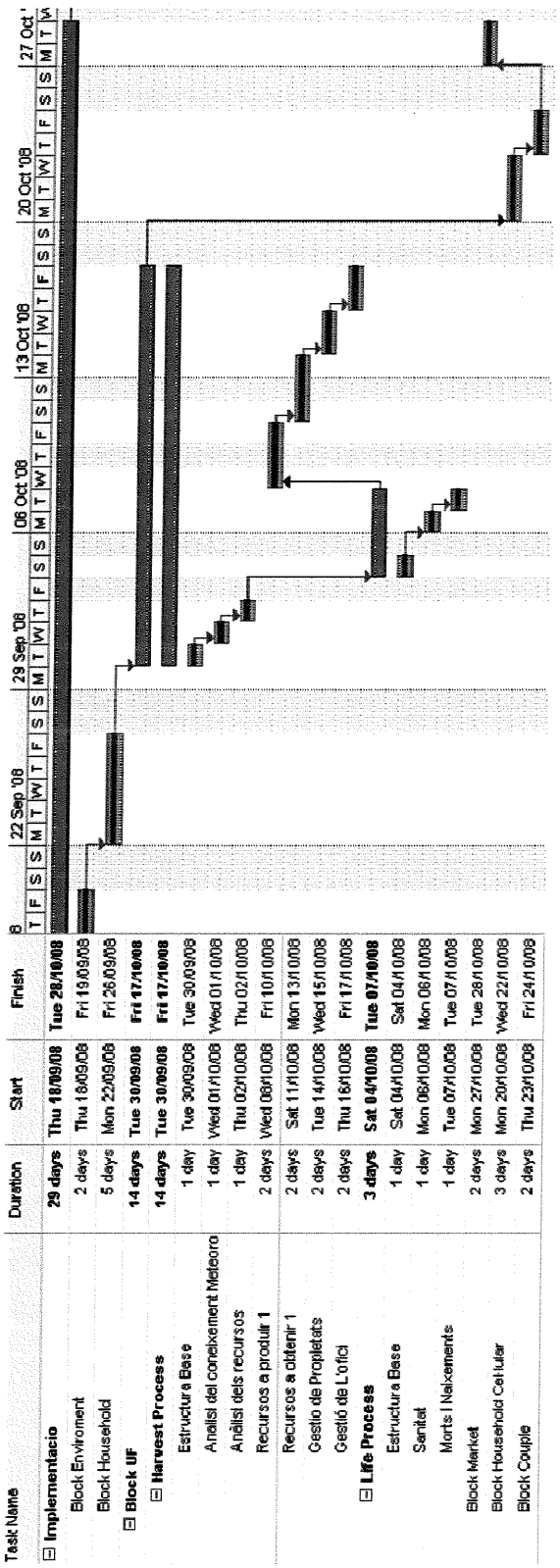
Aquesta fase es dividirà en cinc blocs diferents, cadascun es correspondrà amb la implementació d'un dels blocs que conformen el disseny de l'aplicació (Clima, Unitat Familiar, Parcel·les, Relacions i Mercat). Cadascun dels blocs ha estat planificat segons el nombre d'hores estimat per a la implementació del que representa i es poden consultar al diagrama de Gantt de la evolució del projecte.

8.3.2. Desenvolupament de la fase

La fase d'implementació va transcórrer amb lentitud donada la dificultat tant del programa a crear com del fet de fer servir una eina nova. Va ser la fase que va patir més retards i es va concloure a començaments de novembre.

Al punt de Implementació (pag 76) es detalla aquesta implementació de forma exhaustiva. Per favor referir-se a ell per a més informació.

8.3.3. Calendari



Task Name	Duration	Start	Finish
Implementacio	**29 days**	**Thu 18/09/08**	**Tue 28/10/08**
Block Enviroment	2 days	Thu 18/09/08	Fri 19/09/08
Block Household	5 days	Mon 22/09/08	Fri 26/09/08
Block UF	**14 days**	**Tue 30/09/08**	**Fri 17/10/08**
Harvest Process	**14 days**	**Tue 30/09/08**	**Fri 17/10/08**
Estructura Base	1 day	Tue 30/09/08	Tue 30/09/08
Anellsi del coneixement Meteoro	1 day	Wed 01/10/08	Wed 01/10/08
Anàlisi dels recursos	1 day	Thu 02/10/08	Thu 02/10/08
Recursos a produir 1	2 days	Wed 08/10/08	Fri 10/10/08
Recursos a obtenir 1	2 days	Sat 11/10/08	Mon 13/10/08
Gestio de Propietats	2 days	Tue 14/10/08	Wed 15/10/08
Gestió de L'ofici	2 days	Thu 16/10/08	Fri 17/10/08
Life Process	**3 days**	**Sat 04/10/08**	**Tue 07/10/08**
Estructura Base	1 day	Sat 04/10/08	Sat 04/10/08
Sanitat	1 day	Mon 06/10/08	Mon 06/10/08
Morts I Naixements	1 day	Tue 07/10/08	Tue 07/10/08
Block Market	2 days	Mon 27/10/08	Tue 28/10/08
Block Household Cellular	3 days	Mon 20/10/08	Wed 22/10/08
Block Couple	2 days	Thu 23/10/08	Fri 24/10/08

Tal com s'explica anteriorment, aquesta fase es va dividir segons els 5 blocs que tenia el disseny i cadascun es va subdividir en tasques més petites. La idea de subdividir en tres nivells parteix de la necessitat de poder ajustar les hores de la forma més precisa possible, i això només s'aconsegueix si les tasques són molt simples i fàcils de calcular-ne el cost temporal.

8.4. Tercera Fase: Simulacions i Tests

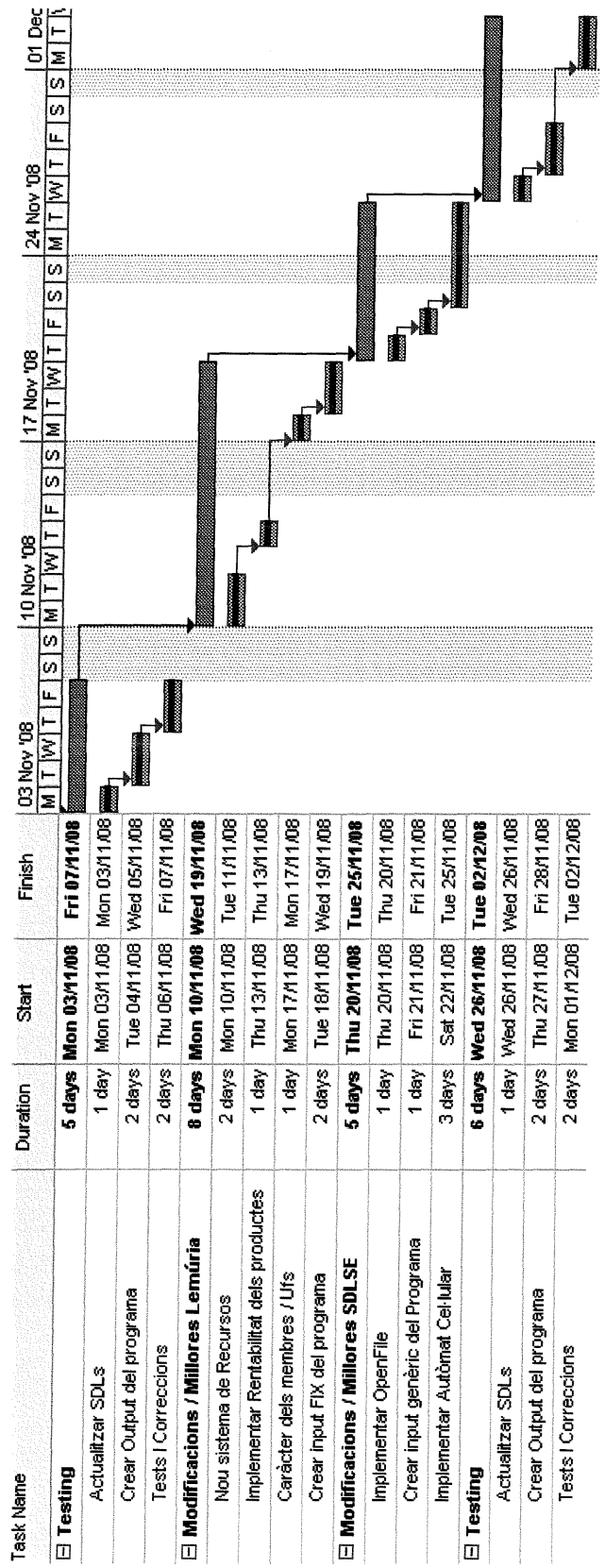
8.4.1. Planificació

Un cop s'havia finalitzat la implementació del programa, s'havia de testejar el correcte funcionament del mateix. Durant la fase d'implementació s'havia, efectivament, testejat alguns aspectes bàsics del mateix, no la forma com podia ser la execució sense errors o la generació correcte de dades. En general, s'havia fet una comprovació de que el programa era robust i que funcionava de forma "sintàcticament" correcta.

No obstant, faltaria veure si el funcionament era coherent i correcte en l'àmbit semàntic. Si les simulacions tenen sentit o no. Aquesta fase es centra en la realització de simulacions i la aplicació de les correccions adequades per a obtenir un funcionament coherent.

Es divideix en dos cicles de test-implementació de millores, cadascun d'ells planejat per a realitzar-se en el transcurs de dues setmanes, una per a testos, una per a la aplicació de les correccions i millores.

8.4.2. Calendari



Task Name	Duration	Start	Finish
Testing	**5 days**	**Mon 03/11/08**	**Fri 07/11/08**
Actualitzar SDLs	1 day	Mon 03/11/08	Mon 03/11/08
Crear Output del programa	2 days	Tue 04/11/08	Wed 05/11/08
Tests I Correccions	2 days	Thu 06/11/08	Fri 07/11/08
Modificacions / Millores Lemúria	**8 days**	**Mon 10/11/08**	**Wed 19/11/08**
Nou sistema de Recursos	2 days	Mon 10/11/08	Tue 11/11/08
Implementar Rentabilitat dels productes	1 day	Thu 13/11/08	Thu 13/11/08
Caràcter dels membres / Ufs	1 day	Mon 17/11/08	Mon 17/11/08
Crear input FIX del programa	2 days	Tue 18/11/08	Wed 19/11/08
Modificacions / Millores SDLSE	**5 days**	**Thu 20/11/08**	**Tue 25/11/08**
Implementar OpenFile	1 day	Thu 20/11/08	Thu 20/11/08
Crear input genèric del Programa	1 day	Fri 21/11/08	Fri 21/11/08
Implementar Autòmat Cel·lular	3 days	Sat 22/11/08	Tue 25/11/08
Testing	**6 days**	**Wed 26/11/08**	**Tue 02/12/08**
Actualitzar SDLs	1 day	Wed 26/11/08	Wed 26/11/08
Crear Output del programa	2 days	Thu 27/11/08	Fri 28/11/08
Tests I Correccions	2 days	Mon 01/12/08	Tue 02/12/08

La fase 3, tal com s'explica al desenvolupament del projecte i al disseny de les fases, es basa en dos cicles de test-correccions. Donat que no es podia preveure quins serien els errors a corregir, es van especificar com a cicles setmanals a emprar de la millor forma possible. És per aquest motiu que cada cicle es divideix en tres fases d'una setmana cadascuna, una de test, una de correccions del Lemúria i una de correccions del SDLPS.

Desenvolupament de la fase

Aquesta fase va transcórrer al llarg del mes de novembre de sense incidents destacables. Es va concloure satisfactòriament i va permetre començar la quarta fase en el dia desitjat.

Per a més informació sobre aquesta fase, referir-se al punt de Simulacions (pag 82), on es troba explicada de forma exhaustiva.

8.5. Quarta Fase: Desenvolupament de la memòria

8.5.1. Planificació

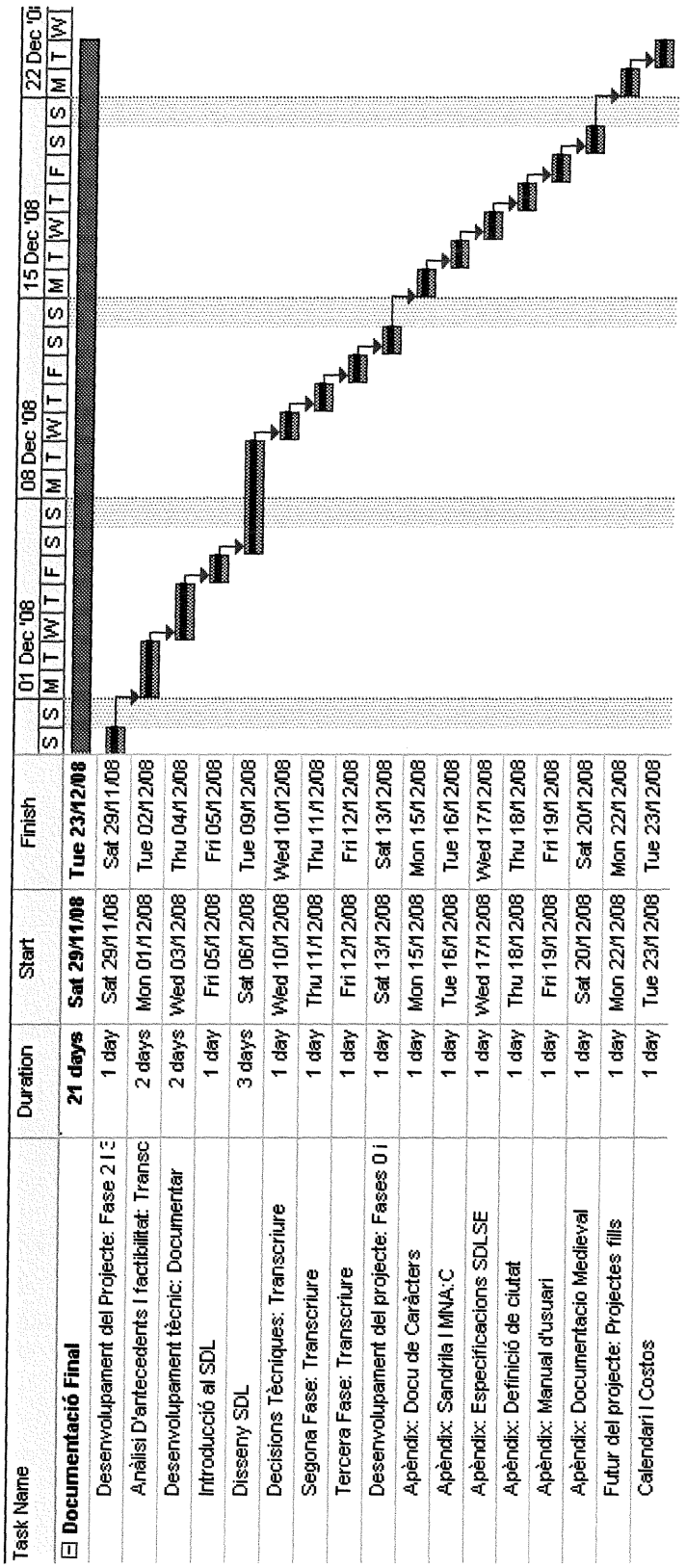
Aquesta fase consisteix en la elaboració d'un document per a resumir el projecte i per a permetre que sigui presentat i avaluat.

Es va planificar que duraria durant el mes de desembre de 2009 ja que, en cas de necessitar més temps per a la implementació / aplicació de testos i millores, es podria recórrer al temps d'aquesta fase.

Està dividida en tasques segons els diferents punts compresos a la memòria.

En aquesta tasca s'han emprat cinc setmanes amb una càrrega de 3 hores diàries de dilluns a divendres, cosa que en resum són 60 hores.

8.5.2. Calendari



Task Name	Duration	Start	Finish
Documentació Final	**21 days**	**Sat 29/11/08**	**Tue 23/12/08**
Desenvolupament del Projecte: Fase 2 i 3	1 day	Sat 29/11/08	Sat 29/11/08
Anàlisi D'antecedents I factibilitat: Transc	2 days	Mon 01/12/08	Tue 02/12/08
Desenvolupament tècnic: Documentar	2 days	Wed 03/12/08	Thu 04/12/08
Introducció al SDL	1 day	Fri 05/12/08	Fri 05/12/08
Disseny SDL	3 days	Sat 06/12/08	Tue 09/12/08
Decisions Tècniques: Transcriure	1 day	Wed 10/12/08	Wed 10/12/08
Segona Fase: Transcriure	1 day	Thu 11/12/08	Thu 11/12/08
Tercera Fase: Transcriure	1 day	Fri 12/12/08	Fri 12/12/08
Desenvolupament del projecte: Fases 0 i	1 day	Sat 13/12/08	Sat 13/12/08
Apèndix: Docu de Caràcters	1 day	Mon 15/12/08	Mon 15/12/08
Apèndix: Sandrila I MNA,C	1 day	Tue 16/12/08	Tue 16/12/08
Apèndix: Especificacions SDLSE	1 day	Wed 17/12/08	Wed 17/12/08
Apèndix: Definició de ciutat	1 day	Thu 18/12/08	Thu 18/12/08
Apèndix: Manual d'usuari	1 day	Fri 19/12/08	Fri 19/12/08
Apèndix: Documentacio Medieval	1 day	Sat 20/12/08	Sat 20/12/08
Futur del projecte: Projectes fills	1 day	Mon 22/12/08	Mon 22/12/08
Calendari I Costos	1 day	Tue 23/12/08	Tue 23/12/08

Aquesta fase, la de desenvolupament de la memòria, és la fase final del projecte. Jo disposava de molta informació recopilada, d'un diari on havia anat anotant a ma els esdeveniments que ocorrien al llarg de tot el projecte i a més a més havia de redactar alguns punts necessaris per a la memòria. Per a aquest motiu, i donat que al desembre esperàvem una allau de feina al lloc on treballo, vaig decidir dedicar-lo exclusivament a la memòria. Es va subdividir la tasca en subtasques d'una, dues o tres tardes de forma que pogués controlar exactament el temps que trigaria i pogués tenir la primera versió de la memòria abans de Nadal.

8.5.3. Desenvolupament de la fase

Un cop finalitzada la implementació i testos, es va passar a elaborar el document final que s'hauria d'entregar. La gran majoria de la informació estava recopilada però mancava unir-la tota en un sol document i a l'hora, transcriure la informació de que disposava sobre conreus que m'havia estat transmesa de forma oral. Es va dur a terme sota els terminis previstos i, un cop finalitzada es va entrar en una etapa de correccions i revisions fins al dia en que havia d'estar lliurada.

9. Implementació

Tal com es comenta amb anterioritat i es pot consultar en el diagrama de Gantt[24], la fase d'implementació comença el dia 18 de setembre.

L'estructura del disseny en blocs pràcticament independents va permetre que es dividís la fase en cinc parts, cadascuna corresponent a un dels blocs.

L'ordre que es va seguir va ser el següent:

1. Bloc Enviroment. Aquest bloc és completament independent dels altres donat que no rep cap informació de la resta de blocs. Això fa que és el bloc més simple de tots i, per tant, la millor elecció per a començar.
2. Bloc Household. Aquest bloc es va triar en segon lloc perquè la seva complexitat és molt petita. Recordem que la seva funció principal és la de recollir les senyals del clima i calcular els recursos produïts cada més. En un futur s'espera que aquest bloc guanyi complexitat i simuli el comportament real de forma més acurada.
3. Bloc Unitat Familiar. Aquest bloc és el més complex de tots, però es va triar en tercer lloc perquè el seu correcte funcionament depèn només dels blocs enviroment i Household i, en canvi, els altres dos li són completament dependents.
4. Bloc de Relacions. Aquest bloc, de la mateixa manera que el següent, depenen de forma quasi exclusiva del Bloc Unitat Familiar. No tenia sentit, per tant, que s'implementessin abans.
5. Bloc de Mercat. És el segon bloc més senzill de tots però depèn dels bloc de Household i Unitat Familiar. Es va deixar per al final perquè era important que els seus antecessors funcionin correctament.

En un primer càlcul, es va estimar que estaria enllestida el dia 15 d'octubre però va resultar que el càlcul era massa optimista per a poder-se portar a terme en el temps proposat.

[24] Consultar secció de Calendari (pàg 70) per a més informació

Bloc d'Enviroment

La principal causa de l'endarreriment es va produir només començar. Aquesta va ser que habituar-se a la eina que es faria servir per a dur a terme les simulacions va resultar més complicat del que en un principi es pensava. Pràcticament la primera setmana de les tres que havia de durar la implementació es va esgotar només intentant de llançar la part més simple del simulador, el procés dins del bloc Enviroment.

Un cop superades les primeres dificultats, el bloc es va desenvolupar sense problemes.

Bloc de Household

Un cop es va tenir enllestit el bloc d'enviroment es va veure que seria impossible tenir el programa acabat a la data prevista, això va portar l'establiment d'una nova data per a la compleció del programa. Aquesta va ser el dia 24 d'Octubre.

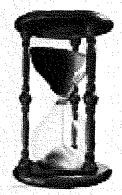
Durant la implementació del bloc de Household es va veure que aquest bloc requeria fer moltes funcions que, implementades en SDL podrien ser excessivament complexes[25]. Això no era un problema a priori donat que el SDLPS permetia implementar funcions en C++. El problema va aparèixer quan es va descobrir que totes les funcions en C++ havien d'estar integrades DINS del propi SDLPS, cosa que feia que la eina de simulació depengués de la simulació executada. Al meu parer, això era intolerable donat que limitava en gran mesura les possibilitats de distribució i execució tant de la eina com de la simulació. Per una banda, l'aplicació SDLPS s'havia de distribuir amb el seu codi si es volia desenvolupar per a ella i, en segon lloc, les simulacions NO es podien executar en qualsevol SDLPS sinó que només es podien llençar sota el que havien estat desenvolupades.

L'estudi en profunditat de la eina va rebel·lar que tota simulació que s'introduïa al SDLPS en un fitxer XML generava un codi C que després es compilava per a generar una biblioteca dinàmica DLL. En aquest punt es va veure que podria ser factible incorporar funcions en C en aquest codi generat. Aquesta solució, però era una feina massa laboriosa donat que cada cop que es modificava el fitxer XML s'havia d'enganxar tot el codi. La solució va venir per l'aïllament i execució fora del SDLPS de les sentències de compilació de la DLL per a

[25] Consultar secció Introducció al SDL (página 21)

permetre la incorporació d'altres fitxers font que continguessin les funcions en C desitjades per a la simulació.

També cal dir en aquest punt que el rendiment del C és creu que de forma genèrica és superior que el del C++. Potser en la majoria d'aplicacions aquesta diferència no es fa palesa, però en aplicacions de simulació, que requereixen grans volums de dades i tenen quantitats ingents de càlculs, aquesta diferència pot ser notable.



En un futur, s'espera que l'SDLPS implementi la opció d'incorporar fitxers externs i que pugui llençar-ne les funcions en C++. Hauria estat una funcionalitat molt útil tot i que la realització d'una modificació així queda fora de l'abast del projecte.

Bloc d'Unitat Familiar

El projecte va anar evolucionant correctament fins el dia 1 d'octubre, en que s'havia de començar la implementació del bloc de Unitat Familiar. En un estudi es va arribar a la conclusió de que la gran complexitat del bloc no es podia plasmar amb facilitat utilitzant funcions simples en C com s'havia fet en el bloc de Household.

El resultat va ser que la informació de les unitats familiars s'emmagatzemés fora de les variables del diagrama SDL. En concret, es van crear una sèrie d'estructures en els fitxers C externs que permetien una forma senzilla de tenir les Unitats Familiars accessibles, estructurades i amb tota la informació disponible en tot moment. A més, amb això es simplificaven els diagrames SDL que, de mantenir tota la informació de la Unitat Familiar passaven a emmagatzemar només el seu identificador i les variables locals que s'utilitzaven puntualment.

La implementació de la Unitat Familiar va funcionar satisfactòriament fins al punt en que s'havia d'implementar la comunicació amb el bloc Household. En fer-ho es va rebel·lar un error important al SDLPS que no tenia en compte la prioritat de les senyals que s'enviaven.

En un principi, el bloc Household tenia 4 estats: CELL, EMPTY, READY i PRODUCING. El primer era un estat artificial que el que feia era recollir l'identificador del Household al que

anava dirigida la senyal entrant, en carregava les dades, es canviava l'estat per a correspondre's amb el del Household determinat i es reenviava la senyal amb l'objectiu de que aquesta caigués en l'estat adequat i es fes la funció pertinent. Aquesta senyal rebotada tenia més prioritat que les que arribaven de la Unitat Familiar i, per tant, havien d'executar-se abans que la resta... però no ho feien. S'executaven totes les de la Unitat Familiar abans i, per tant, el comportament era erroni i caòtic.

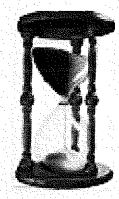
La impossibilitat de corregir l' SDLPS en aquest punt va fer que la solució a aquest problema fos la de suprimir els estats EMPTY, READY i PRODUCING i integrar-ne el comportament dins de l'estat CELL. D'aquesta forma es feia que físicament només quedés l'estat CELL i la resta es codificarien a les dades del propi Household. Amb això es feia que l'estat CELL fes tota la feina seguint l'estat intern del Household. D'aquesta manera les senyal de la Unitat familiar s'interpretaven i s'executaven correctament.

En un futur l'SDLPS implementarà la prioritat de les senyals correctament. No obstant, la dificultat d'aquest punt va fer que fos més factible la modificació de l'SDL enlloc de la reparació de la eina

Un cop la comunicació del Bloc Unitat familiar amb el bloc Household va estar funcionant correctament es va avançar a la creació del procés que controla la salut, morts i naixements dels habitants de la ciutat. Aquest procés es va dur a terme sense problemes tot i que després es va haver de modificar mínimament per a corregir un petit error amb les morts dels membres d'una família.

A continuació es va prosseguir amb la implementació de la Intel·ligència Artificial. Donat que l'objectiu del pfc no és crear una IA completa sinó crear una estructura funcional, es va optar per implementar un algorisme molt senzill que fes les funcions d'intel·ligència artificial.

D'aquesta manera es deixa per a un futur la creació d'una IA més complexa i realista. L'algorisme triat per a fer les funcions de IA va ser el Hill-Climbing[26]



En un futur, si es segueix la construcció de la eina, la IA dels membres serà una xarxa neuronal, que és la forma més aproximada de pensar a les persones que es pot implementar.

En aquest punt va néixer el primer dels "projectes evolució" del Lemúria: La construcció d'una IA de més complexitat mitjançant una xarxa neuronal i amb tècniques més avançades de simulació.

A continuació estava planejada la implementació de la tria dels productes a vendre i comprar de cada família, però es va decidir moure aquest punt fins després de que es tingués implementat el bloc de mercat per tal de poder fer les proves adequadament amb tots els elements disponibles.

Bloc de Relacions

El bloc de relacions havia de ser el quart, i un dels més senzills. Tenia assignats dos dies entre setmana de treball (unes 10-12h), però al implementar-lo es van començar a revelar nombrosos errors dels altres blocs que van fer que s'haguessin de corregir algunes parts del programa. Les parts afectades van ser:

- El bloc d'Unitat Familiar, que no tenia ben implementat el fet de que alguns dels seus membres abandonessin la família per a crear-ne una de nova.

[26] El Hill-Climbing és un algorisme de IA que, donada una solución inicial a un problema, intenta de millorar-la de forma reiterada fins al punt de que ja no la pot millorar més. És un algorisme simple però és molt eficient tant a nivel de procesador com de memoria.

- La IA, que fallava per a les Unitats Familiars de nova creació. Aquest error va portar a descobrir-ne un altre de més greu que es va traduir en la reescriptura del 20% del codi de la IA.
- El procés que controla la mort de les persones, que tenia un bug que feia que els membres que es casaven fossin immortals.

Amb tot això, la compleció del bloc es va concloure en 34h i va portar l'endarreriment de la data de finalització de la implementació al dia 31 d'octubre, data que ja va ser la definitiva.

Bloc de Mercat

El bloc de mercat va ser l'últim dels blocs a ser desenvolupat. Amb molta senzillesa, aquest bloc havia de controlar la compra i venda de productes. La part més complicada del bloc es centrava en el càlcul de preus i en la distribució de les vendes, no obstant, aquesta complexitat estava concentrada sobretot en el codi C, amb el que va portar menys mal de caps que la resta de fases.

Es va concloure per al dia previst i, amb las seva finalització es donava per acabada la fase principal de implementació.

10. Simulacions

Un cop la implementació va estar realitzada, es va passar a una fase de simulacions, correccions i millores.

En aquest punt s'explica com va anar la part de simulacions i quins van ser els esdeveniments i evolució que va tenir.

Nota: En aquest punt es fa referència a simulacions diverses. Per tal de no alterar la llegibilitat del mateix, les dades no es mostren en la forma en que es van obtenir sinó que han estat tractades per a ser mostrades de forma fàcil. Per a consultar les dades originals, referir-se al CD del projecte.

Aquesta fase va començar el dia 3 de Novembre i havia de finalitzar el dia 2 de desembre i tenia per objectiu comprovar el bon funcionament del simulador. Es va dividir en dos cicles de Test-Correccions. A continuació es pot trobar l'explicació detallada de quina va ser l'evolució d'aquesta fase.

Primeres simulacions

Un cop va estar el simulador completament implementat es va procedir a realitzar les primeres simulacions de prova.

L'objectiu d'aquestes simulacions era el de comprovar el bon funcionament dels diversos elements.

L'estudi que es farà serà el següent.

- Estudiar si la IA, que és provisional per a aquest pfc, té les dades necessàries per a funcionar de forma coherent.
- Que els diversos elements funcionen correctament i estan ben cohesionats.

Per tal de poder analitzar els resultats obtinguts per les simulacions, es va implementar un conjunt de funcions que de forma periòdica extreien informació del simulador i la emmagatzemaven en una sèrie de fitxers .csv[27].

Amb l'objectiu de mostrar les simulacions de forma clara i uniforme, totes les simulacions es mostraran seguint l'exemple d'aquesta:

Simulació Test 1.1	**Rurals**	**Urbanes**
Parcel·les	70	30
Unitats Familiars	2	
Anys simulats	50	
Comentaris	La família 1 tenia experiència en el cultiu del blat i la família 2 en tenia en el cultiu de verdures.	
Resultats	A priori, un tendia a pensar que la família 1 es dedicaria al cultiu del blat i la família 2 al cultiu de verdures. No obstant, ambdues famílies es van dedicar al cultiu de verdures. Això es devia a que consideraven més rentable el cultiu de verdures que no pas el cultiu de blat. La conseqüència d'això va ser que el preu dels productes que NO eren la verdura es van anar encarint fins que va arribar un punt en que la família 1 va veure que li seria més rentable cultivar blat i en	

[27] Els fitxers .csv són fitxers de text plans on les dades estan separades per punt I coma (;) o per salt de línea (\n). Són molt pràctics perquè es poden tractar amb eines de full de càlcul com el MS Excel de forma nativa i sense fer cap tipus de conversió o ajust.

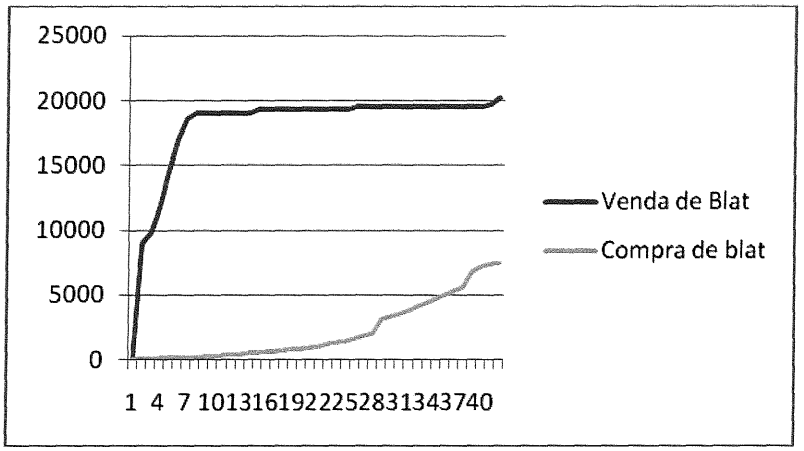
	va començar a produir. Ambdues famílies van tenir descendència i en van aparèixer de noves que es van anar dedicant als diversos productes del mercat.

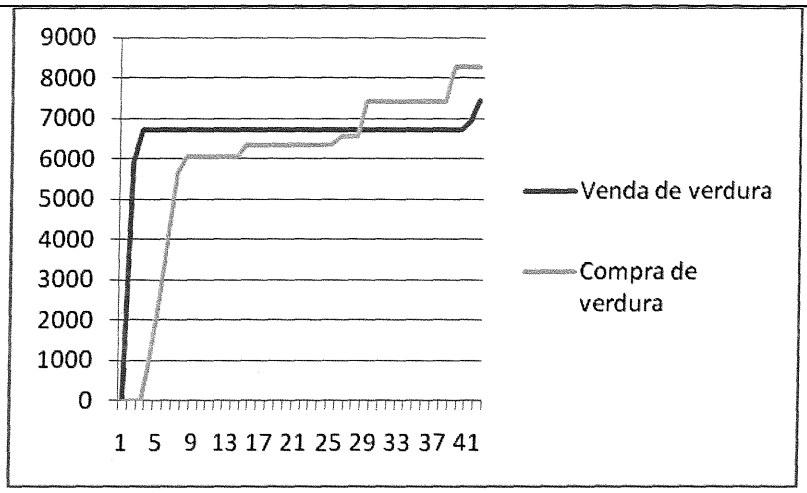
L'anàlisi de memòria utilitzada pel programa va mostrar que el consum era creixent al llarg de tota la simulació. Un càlcul ràpid va portar a la conclusió de que amb la meva màquina, un Core 2 Duo amb 2 Gb de RAM no podria llençar una simulació de més de 100 anys amb aquestes condicions inicials.

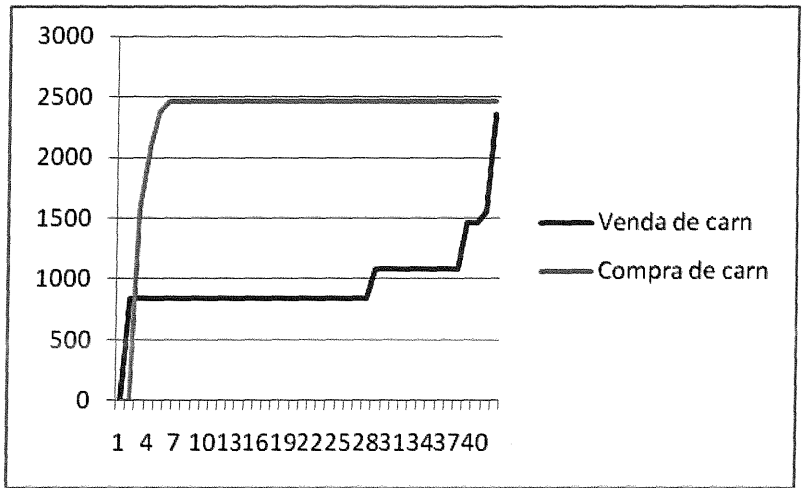
Un cop es va haver analitzat la primera de les simulacions, es va procedir a elaborar-ne una altra que marcaria el punt de partida de la resta de simulacions. En concret es va arribar a la conclusió de que el nombre ideal de famílies seria 6 famílies i el nombre de parcel·les seria de 1000.

Amb aquestes condicions es va llençar la segona simulació...

Simulació Test 1.2	**Rurals**	**Urbanes**
Parcel·les	700	300
Unitats Familiars	6	
Anys simulats	41	
Comentaris	Cadascuna de les famílies estava especialitzada amb l'elaboració d'un producte en concret.	

Resultats	D'aquesta simulació es desprenen algunes conclusions interessants: La primera de totes va ser que les parcel·les tenien una forma de dipositar la informació en el fitxer .csv que no en permetia un estudi fàcil. No obstant, el comportament de les parcel·les era el correcte. La segona conclusió va ser que, donat que els membres de les famílies tenien tendència a casar-se en complir la majoria d'edat, les famílies no es van trobar mai amb la necessitat d'haver de cultivar diverses parcel·les simultàniament. Això va portar que les famílies sempre tinguessin una única parcel·la per a treballar. Un dels punts, però, més interessant va ser l'evolució d'oferta i demanda dels productes alimentaris vitals per a les famílies: Blat, verdura i carn. Donat que les persones que ho cultivaven anaven agafant experiència, cada cop tenien produccions de productes més extenses i, per tant, cada cop hi havia més oferta d'aquests productes per a la mateixa demanda. En concret, l'evolució d'aquests productes va ser la següent: 



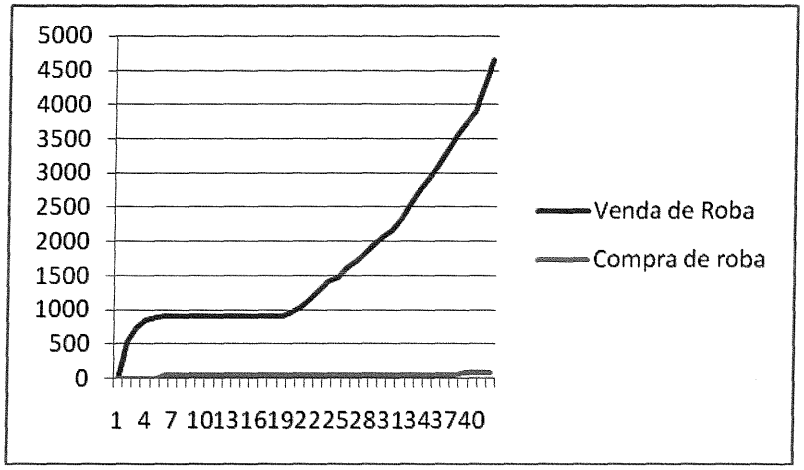


En aquestes gràfiques podem observar com cap al final de la simulació va començar a incrementar-se la compra dels tres productes bàsics. Això es devia, sobretot, al fet de que van començar a aparèixer famílies que es van dedicar a la producció de recursos no alimentaris com la roba, les eines o els elements de luxe. També hi va haver un fort increment de la producció de carn, un aliment que es podria considerar de luxe.

La conclusió que es va poder extreure d'això va ser que la gran oferta de blat en va fer baixar tant el preu que a les famílies no els sortia rentable produir-ne i es van passar a produir altres recursos i a consumir el blat de les altres famílies.

L'altra cara de la moneda la tenen els productes no alimentaris, que van mostrar un error important a la IA de les famílies. La

gràfica següent és prou explicativa...



Les famílies NO tenien en compte que els productes que estaven produint NO s'estaven venent! I els seguien produint! Aquest fet mostrava de forma molt gràfica una incoherència que s'havia de solucionar.

La conclusió de la simulació va ser la següent:

- Els blocs semblen estar cohesionats i la informació circula entre ells de forma coherent.
- A la IA li manca informació del mercat.

Primera fase de correccions i millores

Un cop es van tenir enllestides i analitzades les primeres simulacions, es va procedir a aplicar les correccions i millores que es creien convenients per al simulador. En concret, les correccions que es van aplicar van ser les següents:

1. Corregir el fet de que les famílies venguessin els seus productes encara que ningú els comprés.
2. Corregir els paràmetres d'entrada de la simulació per a potenciar més el comerç (enlloc de donar una gran quantitat de recursos inicialment, donar-ne la quantitat justa per a viure un any i promoure així la creació, compra i venda de recursos.

Les millores van ser les següents:

1. Permetre la parametrització de l'entrada (això incloïa modificar tant el simulador com la pròpia eina, el SDLPS)
2. Implementar les opcions del mn:CA^k:C al SDLPS
3. Aplicar correccions vàries al SDLPS
4. Introduir intel·ligència, caràcter i emprenedoria a les persones.

La primera setmana després de les primeres simulacions es va dedicar a les correccions i millores del simulador. El primer que es va implementar va ser la parametrització del simulador. Aquesta parametrització es va decidir que es basaria en els següents aspectes: Membres, Parcel·les, Recursos, Nutrició, Unitats Familiars i Mercat[28].La parametrització es va produir sense incidents tot i que, de manera provisional i fins que no hi hagués implementada la opció corresponent de càrrega de fitxers dins del SDLPS es va decidir que els fitxers s'adquirissin d'una ubicació fixa.

La segona va ser la correcció de la IA de la unitat familiar. Es va introduir el concepte de "ràtio de venda" de forma que una família podia conèixer si els seus recursos s'havien venut bé o malament a l'any anterior i actuar així en conseqüència.

La introducció d'aquest concepte va fer millorar l'evolució dels mercats i la producció dels diversos recursos.

La tercera va ser la introducció de intel·ligència, caràcter i emprenedoria. Tot i que va costar més del que en un principi es pensava per culpa d'uns errors de corrupció de memòria que es produïen, la implementació es va dur a terme en els marges establerts i el resultat va ser bastant satisfactori.

[28] Consultar secció de Decisions tècniques (pàg 61)

La implementació a nivell tècnic no es volia que fos excessivament complexa i es va intentar de simplificar al màxim tots aquests conceptes. El resultat pràctic va ser que tots els atributs que es volien afegir es van implementar mitjançant nombres reals. Més en concret, es va utilitzar un real per a la intel·ligència, un per al conservadorisme i 4 per al 4 per al caràcter[29].

La setmana següent es va dedicar a la aplicació de millores al SDLPS. La primera que es va aplicar va ser la creació d'una funció que permetés a la simulació demanar a l'usuari la introducció d'un fitxer per a poder parametritzar-se. Aquesta funció es va implementar sense problemes. Un cop va estar llesta, es va lligar la parametrització del programa, que recordem que utilitzava els fitxers d'una ubicació fixa fins ara, per a poder adquirir un fitxer d'entrada de qualsevol part del sistema de fitxers.

La segona millora que es va aplicar va ser la opció de permetre al SDLPS de compilar automàticament els fitxers .c que es volguessin integrar dins de la simulació.

Recordem que, fins a aquest moment, s'estava utilitzant un programa extern al SDLPS per a fer la compilació de la biblioteca DLL on hi havia la simulació (XML).

El procediment que es va seguir per a la incorporació d'aquesta funcionalitat va ser el següent:

- Si es vol que s'integri codi extern al DLL compilat, els fitxers .c i .h s'han de col·locar en una carpeta anomenada SDLCodeExternal a la mateixa ubicació que el fitxer XML de la simulació.
- A més a més, ha d'existir un fitxer SDLCodeExternal.h que sigui el que contingui els includes i funcions que es cridaran des de la simulació.

En un futur, si es segueix la construcció de la eina, es podrà configurar de quina carpeta s'ha d'extreure el codi extern i quina és la seva capçalera bàsica.

[29] Segons un estudi publicat per Katharine Cook Briggs i Isabel Briggs Myers al 1923, tot caràcter es pot representar a partir de 4 valors. Consultar secció El MBTI (Myers-Briggs Type Indicator) (pàg 124).

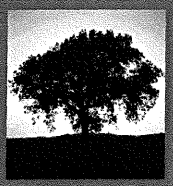
La tercera i última millors que es va implementar al SDLPS va ser la opció de crear un MNA:C. Aquesta millora, en teoria, havia de permetre la aplicació d'un model meteorològic i de producció de recursos molt realista.

La base d'aquesta millora es basava en dues funcions: MNACSetState i MNACGetState. Aquestes dues funcions tenien la responsabilitat d'emmagatzemar i carregar l'estat de les diverses cel·les segons la seva posició i capa.

L'especificació d'ambdues funcions ja estava creada i es va procedir a la seva implementació.

Un cop van estar implementades, vaig descobrir amb bastant horror que la seva aplicació al simulador de ciutats (que ja implementava mitjançant el codi extern aquestes funcionalitats) implicava canvis massius en el codi. El primer canvi massiu era que s'havia de crear una capa per cada atribut. El simulador emmagatzema de cada parcel·la de l'ordre de 10 atributs, amb el que s'havien de crear 9 capes extra i s'havien de programar una a una. A més, hi havia també un problema important amb el tipus de dades que s'emmagatzemaven en aquestes dues funcions. Ambdues funcionaven amb nombres reals mentre que a l'aplicació, la majoria de les dades de les parcel·les eren enteres. Pot no semblar un problema, però recordem que hi valors enters que no es poden representar en coma flotant i jo necessitava aquests enters amb tota la seva precisió (molts d'ells servien per a referenciar posicions a matrius).

Donat que ens trobàvem a mitjans de novembre, aquests canvis no es podien aplicar per la manca de temps i es va optar per a descartar l'aplicació d'aquesta millora per al futur del projecte.



En aquest punt va néixer el segon dels "projectes evolució" del Lemúria: La construcció d'un sistema de parcel·les molt més realista i que implementi el MNA:C.

Segones Simulacions

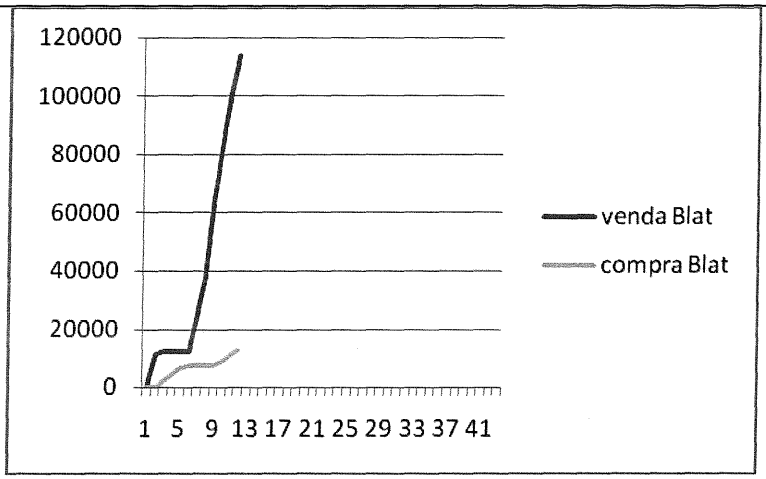
Un cop finalitzades les primeres correccions es va optar per a realitzar més simulacions. Aquest cop s'havia de provar el correcte funcionament de la parametrització, de la compilació

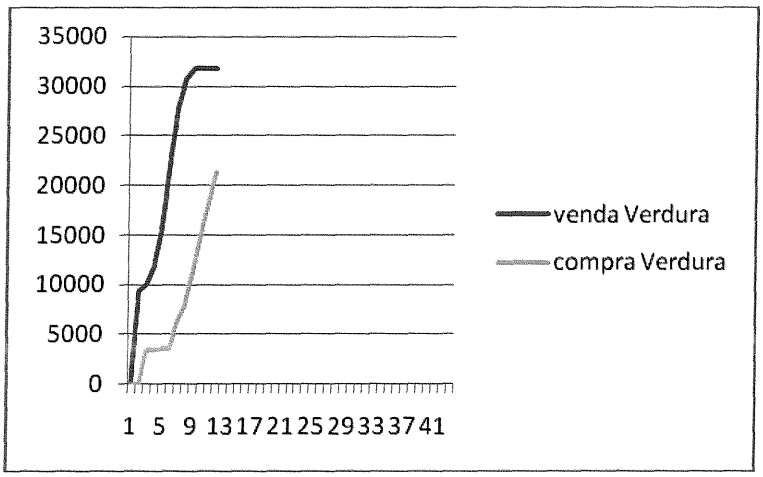
automàtica del SDLPS i que ja no es repetien el problemes que s'havien mostrat en les simulacions anteriors.

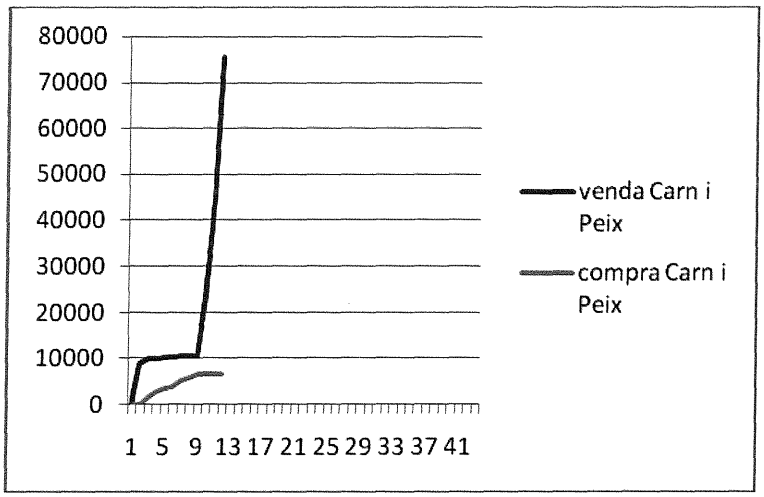
El primer test que es va fer va ser la compilació. Es va compilar el simulador , per una banda, mitjançant el programa extern i per l'altra mitjançant la nova funcionalitat del SDLPS. Es va testejar el comportament en ambdós casos i els resultats generats van ser els mateixos. Tot i ser esperançador el fet de que els dos generessin els mateixos resultats, es va utilitzar una eina per a trobar diferències de fitxers i es va aplicar sobre els les dues DLLs, ambdues eren idèntiques. Feta aquesta prova, es va concloure que la compilació automàtica funcionava correctament.

Després es van llençar una sèrie de simulacions utilitzant dades molt semblants a les emprades en la primera fase de tests. El resultat va ser el següent:

Simulació Test 2.1	**Rurals**	**Urbanes**
Parcel·les	700	300
Unitats Familiars	6	
Anys simulats	41	
Comentaris	Cadascuna de les famílies estava especialitzada amb l'elaboració d'un producte en concret.	
Resultats	Es van llençar les simulacions i el resultat de forma gràfica entre la compra i venda de productes va ser la següent:	

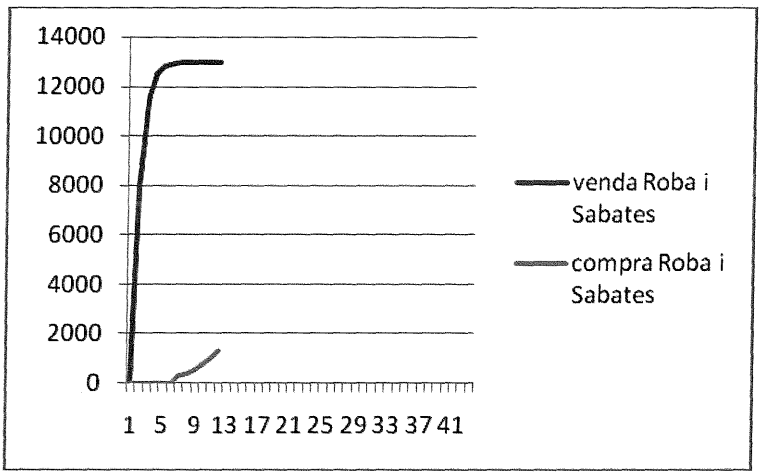






Hem de comentar que l'any 9 es va incorporar una nova família a la producció de carn que era molt intel·ligent. Per aquest motiu van començar a aprendre molt ràpid i a obtenir millors resultats cada any.

La producció de roba, que tan mals resultats havia donat en els testos anteriors, va donar el següent resultat:



Una família intel·ligent havia començat la producció de roba i eren els únics proveïdors d'aquest recurs. Tot i que el preu baixava per manca de demanda, els era suficient per a mantenir-se.

La primera idea que podem treure d'aquí és que les famílies ara semblaven més conscients del mercat i produïen de forma més adequada a ell tot i que es van treure les següents conclusions:

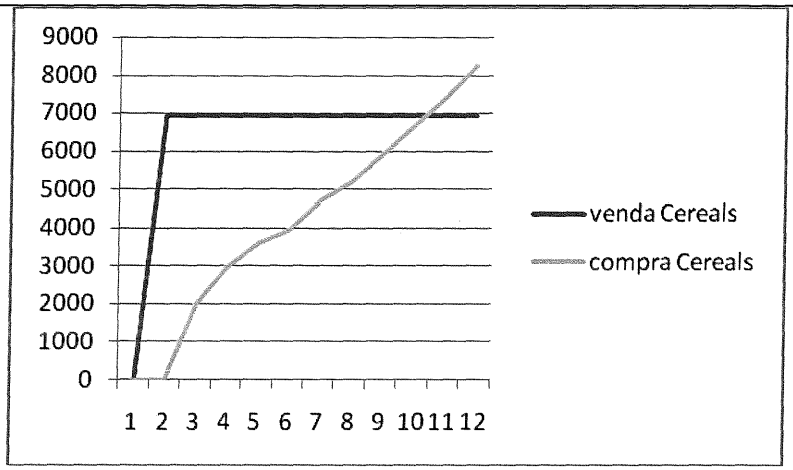
- La compravenda de recursos no era correcte. Les possibles causes podien ser:
  - L'aprenentatge de les famílies era massa ràpid
  - Les parcel·les semblaven produir massa recursos
- La connexió entre els diversos elements era correcte

Es van comprovar els recursos produïts per les parcel·les eren correctes comparats amb les dades de que disposava. També podia ser un problema del consum, però això també es va comprovar i va resultar que el consum era també correcte. La conclusió final va ser que aquest problema amb els recursos es devia a la poca quantitat de gent que es simulava. Si tenim en compte que les ciutats que es volen arribar a simular poden tenir entre 1000 i 10000 famílies, aleshores podem deduir que l'aparició d'un sector dedicat a la producció de productes "urbans" farà que s'equilibrin tots els aspectes incoherents d'aquesta simulació.

A continuació es va voler testejar l'entrada de dades. Es van realitzar mini-testos modificant els paràmetres d'entrada i veien si es reflectien en la primera sortida d'informació. Tots van funcionar correctament.

A continuació es va passar a testejar si es reflectia l'entrada de nous tipus de productes i de nous tipus de parcel·les. Es va realitzar el següent test:

Simulació Test 2.2	**Rurals**	**Urbanes**
Parcel·les	700	300
Unitats Familiars	6	
Anys simulats	20 (Una simulació curta)	
Comentaris	- Cadascuna de les famílies estava especialitzada amb l'elaboració d'un producte en concret. - Es va dividir el blat en dos productes diferents: "Blat" i "Altres Cereals". Una de les famílies es va especialitzar en el cultiu de cereals per a propiciar l'aparició de la producció d'aquest producte.	
Resultats	Els cereals van ser integrats dins de la simulació i la família "especialista" va dedicar-se a produir-los. A continuació es pot veure la compravenda de cereals:	



Com es pot observar, la venda de cereals és constants donat que la família especialista ja dominava el conreu de cereals. La compra va anar pujant a mesura que creixia la població amb els nous naixements.

D'aquí podem concloure que els productes nous s'integren de forma coherent i les famílies els tenen en compte a tots els efectes al llarg de la seva vida.

Segona fase de correccions i millores

Aquesta fase, pensada per a corregir els possibles errors que podia haver revelat la fase de test es va reorientar completament.

Per un cantó, les simulacions eren coherents i, a banda de correccions menors, no s'havia d'aplicar cap millora.

Per l'altre, com que el SDLPS és un projecte en desenvolupament que està essent construït per tres persones simultàniament, es van incorporar al projecte unes modificacions al SDLPS que van esborrar algunes millores que jo havia introduït. Això va fer que una part del simulador deixés de funcionar. Aquesta setmana es va dedicar, sobretot, a la correcció d'aquests errors per tal de que la simulació tornés a funcionar correctament.

10.1. Conclusions de les simulacions

Tot i que el resultat pot semblar que sigui poc coherent en alguns punts, cal comentar el fet de que el simulador s'ha mostrat robust i que tots els seus components han funcionat de la forma que era esperada. Cal notar també que, com era d'esperar, el comportament dels agents és simple donat que la seva IA és provisional i que caldrà augmentar la complexitat del comportament de les parcel·les.

En resum, els objectius de crear un prototip que permeti la recreació bàsica i primera de societats han estat assolits. Això, no obstant, no vol dir que el projecte hagi estat finalitzat, al Lemúria li queda un camí molt llarg per recórrer que es farà un cop finalitzat aquest projecte.

11. Costos del projecte

El desenvolupament del Lemúria ha estat una tasca difícil, i com tota tasca ha tingut una dedicació de moltes hores que va ser planificada amb antelació i que ha representat uns costos associats al material utilitzat i a les hores emprades.

Els costos del projecte han estat centrats, bàsicament, en la utilització de material i en l'aplicació d'hores de treball. Tota la informació referent als costos es pot consultar al quadre següent:

11.1.1. Costos per Hores

Fase	Hores Setmanals	Hores totals	Preu per hora	Cost Econòmic
Documentació	4	48	7	336
Disseny	15	225	40	9000
Implementació	32	352	10	3520
Tests i Correccions	20	80	10	800
Memòria	15	60	7	420
Presentació	-	20	7	140
Total Hores		**785h**		**14216€**

11.1.2. Costos de software

Software	Cost (€)
Microsoft Windows XP	192
Microsoft Office 2008	107
Microsoft Visual Studio 2008	573
Sandrila	23
Total Software	**895€**

11.1.3. Cost total:

En resum, el cost total del Lemúria ha estat de **785 hores i de 15111€.**

12. Treball futur

Tal com s'ha comentat al llarg de tota la documentació, el Lemúria presentat no te vocació de ser un projecte acabat. Al contrari, el Lemúria s'ha plantejat en tot moment com el primer prototip d'una futura eina dedicada a la simulació social de les ciutats.

Partint d'aquesta premissa, ens trobem en que el Lemúria ha d'evolucionar si vol arribar a ser plenament funcional. Amb aquesta idea neixen els tres projectes futurs que s'haurien de fer per a que la eina arribés a ser funcional.

12.1. Projecte Lemúria IA

Aquest projecte consisteix en la elaboració d'una IA per als membres i les unitats familiars mitjançant una xarxa neuronal o algun algorisme de gran complexitat. Tal com s'ha comentat, dins d'aquest projecte s'ha implantat un algorisme molt simple de IA, i aquest ha de ser substituït per un altre de més sofisticat.

Aquest projecte probablement tindria una mida semblant a la de un PFC sencer.

12.2. Projecte Lemúria Terrain

En aquest projecte es fa servir una simplificació del funcionament dels camps i de les plantes que s'hi cultiven. Seria molt interessant disposar d'un sistema de clima i conreus que fos més realista del que actualment es disposa.

El projecte partiria de la aplicació en profunditat d'un MNA:C, un sistema de simulació basat en una superfície on les diverses cel·les d'un grid interaccionen entre elles i amb condicions externes.

També dins d'aquest projecte s'inclouria la millora del sistema de clima, que és, de moment, rígid tot i que ja aconsegueix alt grau de realisme.

13. Manuals d'Usuari

13.1. Manual del SDLPS

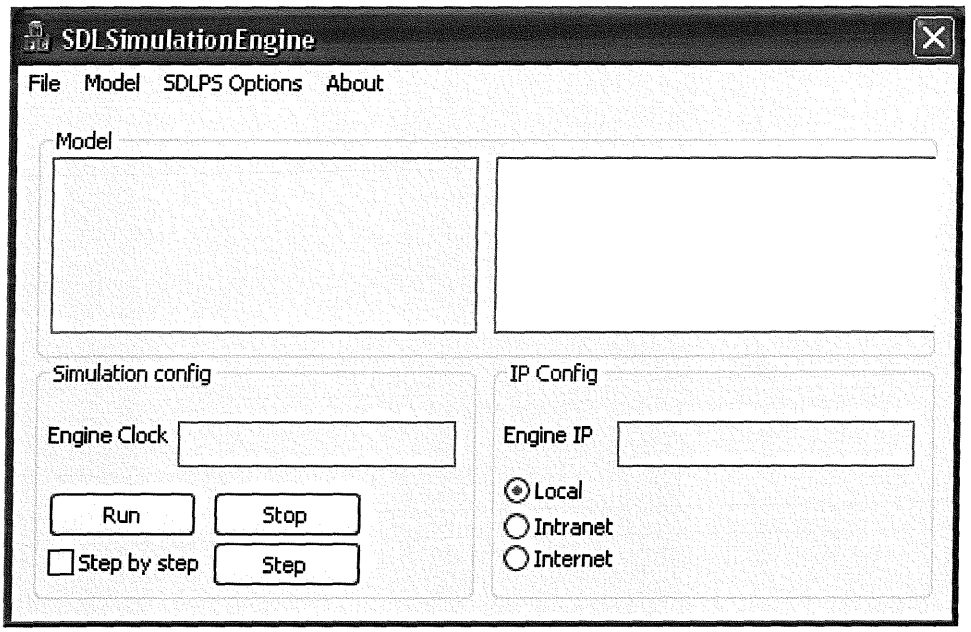
El Lemúria és una simulació que funciona, tal com s'ha explicat amb anterioritat, sota l'entorn generat pel SDLPS. Per tant, per a poder executar-lo, caldrà primer instal·lar i executar el SDLPS.

Cal notar que aquest manual fa referència a la versió Beta del SDLPS, amb el que és probable que es modifiqui de cara a un futur.

És important notar aquí que la eina permet, per una banda compilar la simulació si se'ns proporcionen els codis font i el fitxer XML corresponents. També permetrà llençar-la si se'ns proporciona l' XML i el fitxer DLL corresponent.[30]

Quan obrim el SDLPS veurem una finestra com la següent:

[30] Veure Especificacions del SDLSimulationEngine

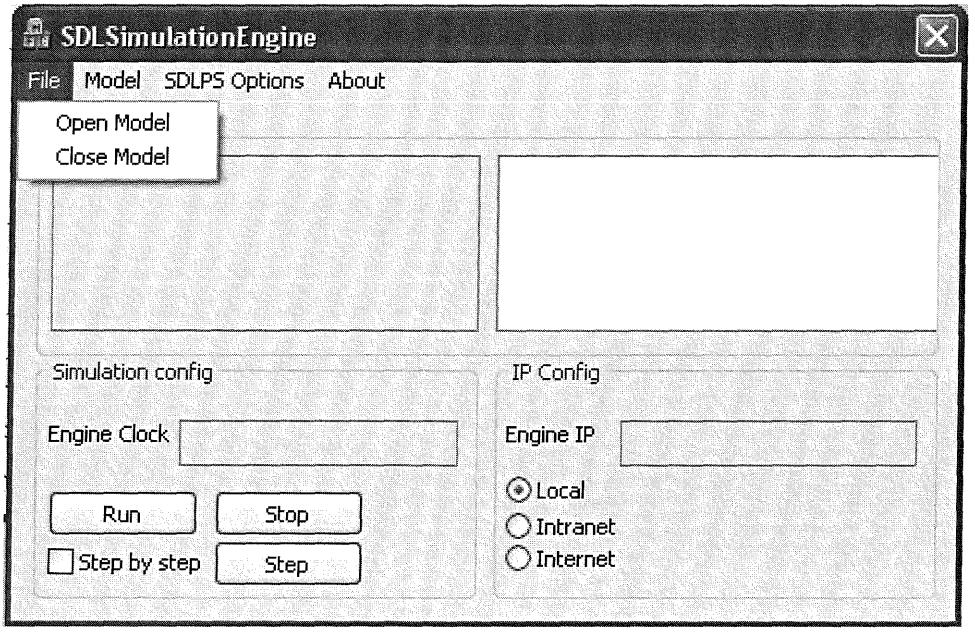


En aquesta finestra hi podem distingir una sèrie d'elements que són els següents:

1. Finestra con es mostrarà el codi XML de les simulacions
2. Finestra on es mostrarà l'arbre que representa el codi XML de la simulació
3. Botó de començar una simulació
4. Botó de avançar un pas en la simulació
5. Botó per a aturar la simulació
6. Checkbox per a marcar si es vol una simulació pas a pas o continuada.
7. Menús

13.1.1. Menú Fitxer

Dins del menú de fitxer tenim les opcions de carregar i descarregar una simulació.



Carregar (Open Model)

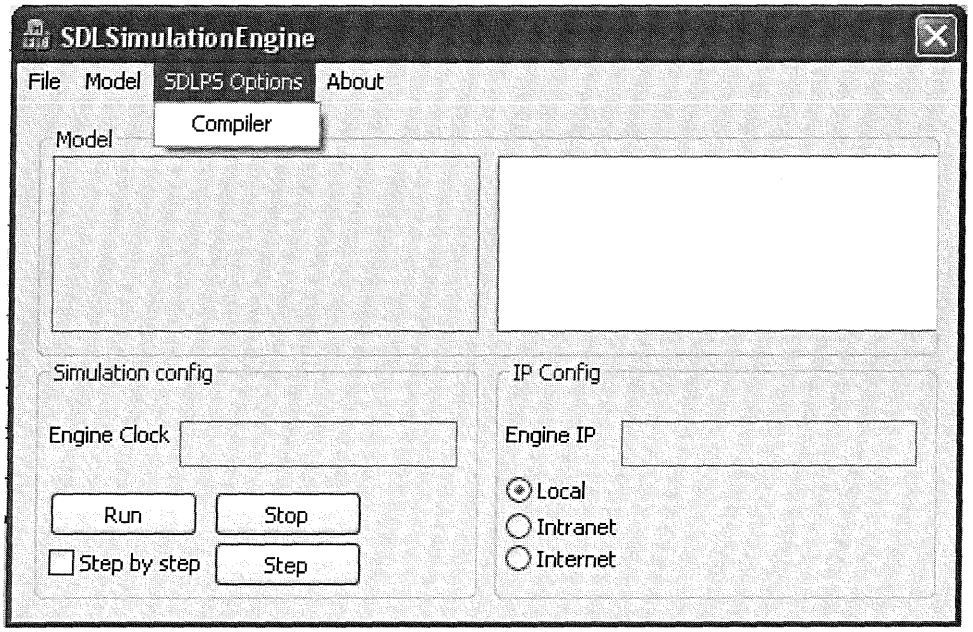
Si clickem sobre la opció de carregar una simulació, se'ns mostrarà una finestra que ens deixarà seleccionar l'arxiu XML on està emmagatzemada la simulació que volem llençar.

Si triem un arxiu XML, aquest serà carregat i es mostrarà el seu contingut tant a la finestra 1 com 2 (veure figura anterior). A la finestra 1 es mostrarà el contingut del XML en forma de text pla i en la finestra 2 es mostrarà el mateix document XML però en forma d'arbre.

Tancar simulació (Close Model)

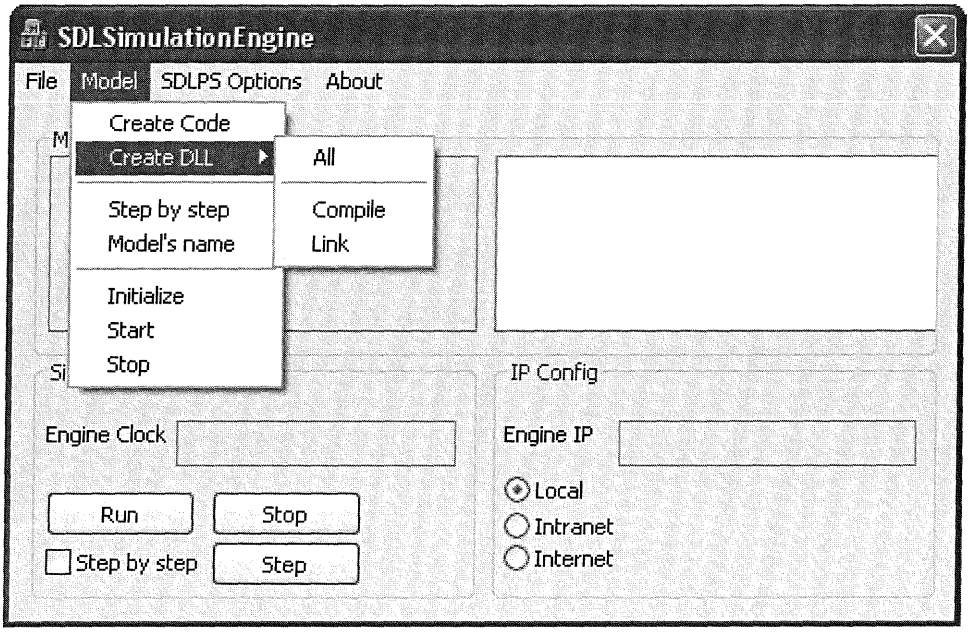
Si tenim una simulació carretada, podrem tancar-la si clickem aquesta opció.

13.1.2. SDLPS Options



En el cas de que vulguem compilar una simulació, primer hem de seleccionar quin compilador fem servir. Aquest menú només té una opció que ens demanarà la ubicació del fitxer gcc.exe del compilador que vulguem utilitzar.

13.1.3. Menú Model



Create DLL / Create Code

Aquest menú és el que ens permetrà controlar la simulació. Si la simulació s'ha de compilar i ja hem seleccionat el nostre compilador, aleshores podem seleccionar la opció de compilar o Linkar la nostra simulació.

Tal com s'especifica, si es disposa de fitxers de codi extern, aquests han d'estar localitzats dins de la carpeta SDLCodeExternal i aquesta ha de contenir un fitxer anomenat SDLCodeExternal.h on es concentrin els includes en C que necessita el XML (el codi XML farà crides en C, aquestes i totes les estructures que utilitzi han d'estar incloses en el SDLCodeExternal.h).

Initialize

Abans de començar qualsevol simulació, aquesta s'ha d'inicialitzar. Recordem que els diagrames SDL partien d'un estat inicial al que no tornarien, executaven una sèrie d'accions i es situaven a un dels seus estats de treball. Quan fem clic a Initialize, si la simulació ha estat compilada correctament, s'efectuarà aquest moviment i les operacions corresponents.

És probable que si la simulació requereix d'algun fitxer extern, ens el demani en aquest punt.

Step by Step

Aquesta opció determinarà si la execució de la simulació es farà step by Step, és a dir, pas per pas, que s'aturarà a cada increment del temps que es faci, o si s'efectuarà de forma continuada fins que l'usuari triï la opció Stop.

La majoria de simulacions simples poden funcionar step by step. No obstant, si es vol executar una simulació de complexitat mitra o gran, l' Step by Step es fa inútil i només fa que s'alenteixi el procés de simulació.

Start

Aquesta opció farà que la simulació comenci a executar-se. Si es tracta d'una simulació Step by Step, aleshores realitzarà un únic pas. Si no és el cas, aleshores començarà a simular de forma ininterrompuda fins que l'usuari clic sobre el botó Stop.

Stop

Aquesta opció aturarà la simulació en cas de ser una simulació sense Step by Step. Un cop clickada aquesta opció, la simulació s'atura i només es tornarà a executar quan es clicki sobre Start.

13.2. Manual d'Usuari del Lemúria.

El Lemúria, al simular ciutats, és una simulació que requereix una forta parametrització. És, per tant, molt important que la parametrització el simulador sigui el més flexible possible i, per tant, abans de començar a executar-la, caldrà preparar una sèrie de fitxers amb la informació necessària.[31]

13.2.1. Fitxers de parametrització

Per a parametritzar el Lemúria necessitarem introduir TOTS els següents fitxers ubicats. És necessari que tots estiguin a la mateixa carpeta, donat que el simulador només ens preguntarà la localització del primer i deduirà que tots estan al mateix lloc.

Tots són fitxers csv que es poden modificar utilitzant qualsevol eina d'edició de text o edició de fulls de càlcul. La primera fila de tots ells està pensada per a contenir el títol de cadascuna de les columnes i facilitar-ne així la introducció de dades. Qualsevol informació a la primera fila serà obviada pel simulador.

Lemuria.csv

En aquest fitxer s'hi introduirà informació bàsica de la parametrització. No disposa de cap format específica tot i que es suggereix aquí escriure el nom de la simulació, l'autor i qualsevol comentari que es cregui pertinent.

[31] Per a més informació sobre la informació necessària per a definir i parametritzar una ciutat, consultar l'annex VI: Definició de Ciutat

Household_types.csv

Aquest fitxer disposa d'un format com el següent:

Id	Tipus
0	Rural
1	Urbà

Consta de dues columnes i una sèrie de files. La primera fila conté els títols de les columnes i serà obviada quan es carregui la informació al simulador.

La primera columna ens indicarà l'identificador que tindran cadascun dels tipus de Households (parcel·les) que utilitzarem. A la segona columna s'hi podran veure els noms d'aquests households.

Product_Types.csv

Aquest fitxer conté informació dels diversos productes amb els que tractarem a la simulació. De la mateixa forma que passava en el fitxer anterior, la primera fila conté els noms de les columnes i la resta tindran, cadascuna, informació sobre un tipus de producte determinat. La informació que caldrà omplir de cada producte és la següent (de forma ordenada com es requereix per al simulador):

Id: Identificador del producte

Nom: Nom del producte

Tipus de Household: id del household on es pot cultivar

Preu per defecte: En tota simulació, els productes partiran d'un preu de mercat que després anirà evolucionant. Aquí hem de marcar quin serà el preu inicial.

Kcal, greix, Vitamines i Minerals: Quantitats en Kcal o en grams de l'aport nutricional que té aquest producte si és de menjar. En cas de no ser un producte alimentari, aquestes columnes s'obviaran.

Es aliment?: Un 1 en aquesta columna indicarà si el producte és un aliment. Un 0 voldrà dir que no

Felicitat: Felicitat que aporta el consumir el producte. Un nombre enter on el mínim és 0 que ens indica la felicitat que es rep per cada unitat del producte consumida.

Mes de recol·lecta: Valor entre -1 i 11 que indica en quin mes es recull la producció d'aquest producte. Si el valor és -1, vol dir que el producte es pot recollir cada mes.

Producció Base: Unitats de recursos produïdes del producte com a mínim per una sola persona.

Susceptible al clima: Aquesta columna tindrà un 1 si aquest producte és susceptible a les condicions atmosfèriques o un 0 si no.

Jan-Temp, Jan-Pluja, Jan-Vent, Feb-Temp,... Quantitats ideals de temperatura (en graus centígrads), Pluja (en cl/m^2) i Vent (Valor entre 1 i 5 on 1 és poc vent i 5 molt vent) per a cadascun dels 12 mesos de l'any. Si el producte no és susceptible al clima, s'obviaran.

Consum.csv

Aquest fitxer contindrà el consum mig de cadascun dels productes introduït per a la ciutat que es simula. Aquest consum estarà dividit en tres franges d'edat: Infants, Adolescents i Adults. Les columnes que caldrà omplir són:

Id del producte: identificador del producte al qual farem referència

Consum Adult: Unitats de recursos que consumirà un adult del producte determinat en el període d'un any.

Consum Adolescent: Unitats de recursos que consumirà un adolescent del producte en el període d'un any.

Consum Infantil: Unitats de recursos que consumirà un infant del producte en el període d'un any.

Unitats_familiars.csv

Aquest fitxer contindrà informació dels recursos amb els partiran cadascuna de les unitats familiars del simulador. Les columnes seran les següents:

Id: Identificador de la unitat familiar.

<id producte>: A partir de las segona columna, cadascuna de les següents farà referència a un dels productes introduïts amb anterioritat. És important que estiguin ordenats. El valor introduït indicarà la quantitat en estoc que tindrà la família del producte determinat.

Membres.csv

Aquest fitxer contindrà informació dels membres que començaran la simulació. La informació que caldrà donar de cadascun és la següent:

Id: identificador del membre

Unitat_familiar: Identificador de la unitat familiar a la que pertanyen.

Sexe: 0 si és un home, 1 si és una dona

Nom, Cognom1 i Cognom2: Nom i cognoms del membre.

Edat: edat del membre

És cap de família?: Aquest camp tindrà un 1 si la persona és un dels dos caps de família de la unitat familiar. El rol de cap de família correspon sempre a un home i una dona que són els que poden tenir fills que s'introdueixin a la família. Cap altre membre no pot tenir fills si no és cap de família.

<id producte>: De la mateixa forma que abans indicàvem l'estoc, aquí indiquem quina serà la quantitat d'experiència que tindrà una persona en produir un producte determinat. Aquest valor anirà entre 0 i 100.

Household.csv

Aquest fitxer té informació que serà obviada tant a la primera fila (com tota la resta de fitxers) com a la primera columna. La idea és que aquest fitxer representarà el grid de 100 x 100 caselles dels diversos households indicant-ne el tipus de que són. (Per tant, omplirem fins a la fila 101 i la columna 101).

13.2.2. Executant el Lemúria.

Un cop haguem carregat el fitxer XML de la simulació i li haguem donat a Initialize, el programa ens demanarà un fitxer csv. Hem de triar el fitxer lemuria.csv, on hi ha la informació bàsica del simulador.

Si la càrrega s'efectua correctament, aleshores ja es podrà seleccionar la opció d'Start i la simulació es començarà a executar.

A mesura que aquesta simulació s'executa, anirà generant els fitxers de sortida. Quan clickem sobre Stop, aquests fitxers passaran a estar disponibles per a ser llegits i interpretats.

14. Bibliografia

Bejar Alonso, J. (sense data). *Inteligencia Artificial.* Recollit de http://www.lsi.upc.edu/~bejar/ia/ia.html

Consumo de Caloráis necesario. (sense data). Recollit de http://alimentacion.interbusca.com/herramientas/consumo-calorias-necesario/

Fonseca i Casas, P. (sense data). *Pau Fonseca i Casas.* Recollit de http://147.83.24.206/~pau/

Garcia Sebastián., M., Gatell Arimont, C., Llorens Serrano, M., Pons Granja, M., Ortega Canadell, R., Roig Obiol, J., et al. (2005). *Marca 2.* Barcelona: Vicens Vives.

Life expectancy. (sense data). Consultat el 21 / Abril / 2008, a http://en.wikipedia.org/wiki/Life_expectancy

Medieval demography. (sense data). Consultat el 2008 / Abril / 21, a http://en.wikipedia.org/wiki/Medieval_demography

Medieval Household. (sense data). Consultat el 21 / Abril / 2008, a http://en.wikipedia.org/wiki/Medieval_household

Myers-Briggs Type Indicator. (sense data). Recollit de http://en.wikipedia.org/wiki/MBTI

Necesidades Nutrcionales del Organismo. (sense data). Recollit de http://www.zonadiet.com/nutricion/necesidad.htm

Protagonistas de la historia. (sense data). Recollit de http://www.artehistoria.jcyl.es/historia/contextos.htm

Ruano, E. B. (sense data). *La historia de la vida cotidiana en la historia de la sociedad Medieval.* Recollit de http://www.vallenajerilla.com/berceo/benitoruano/historiadelavidacotidiana.htm

Tablas de Aporte de los alimentos y equlibrio Nutricional. (sense data). Recollit de http://www.zonadiet.com/tablas/

Time traveler's guide. (sense data). Consultat el 2008 / Abril / 21, a http://www.channel4.com/history/microsites/H/history/guide12/part10.html

Vida cotidiana en Plena y baja Edad Media. (sense data). Consultat el 2008 / Abril / 21, a http://www.artehistoria.jcyl.es/historia/contextos/1334.htm

Annexos

1. Sobre la Societat Medieval

Nota: Aquest Annex és una transcripció dels apunts de la societat medieval que vaig anar recollint en la meva recerca.

1.1. Estaments

La societat medieval estava estructurada en diversos estaments que determinaven la vida de les persones. En concret existien 3 estaments:

Els privilegiats: Formats per la noblesa i pel clergue. Eren els que tenien el poder a l'edat mitja i governaven per sobre dels no privilegiats.

La noblesa: grup de persones que controlaven el poder polític a l'edat mitjana. Tenien el poder militar.

El clergat: Grup de persones de vocació religiosa que tenien poder sobre la fe de la resta. El seu poder polític era notable tot i que no arribava al nivell de la noblesa.

Els no privilegiats: Formats pels serfs i els vilans.

Els Vilans: Eren persones lliures (no estaven sota el poder directe d'un noble o d'un clergue). No eren propietaris de les terres que treballaven. En aquest grup entren els burgesos, artesans,...

Els Serfs: Eren persones sotmeses al poder d'un noble a través del vassallatge. Conreaven la terra i havien de demanar permís al senyor per a fer qualsevol cosa.

Els Esclaus: Persones normalment estrangeres (musulmans, africans,...) que feien de criats a les cases i treballaven al camp, les pedreres,...

1.1.1. Altres dades:

Expectativa de vida: 20 – 30 anys. Sent més curta per als privilegiats, que basaven la seva alimentació en productes basats en carn i més llarga per als no privilegiats, que duien una alimentació més equilibrada.

Dots i matrimoni: Quan una noia es casava, totes les seves possessions passaven a ser del marit. L'edat mínima per a casar-se era de 12 anys per a les noies i de 14 per als nois. Els nobles no regien amb aquesta regla i es podien casar a qualsevol edat (el cas pitjor és el del Príncep David d'Escòcia, que el van casar al 1328 quan tenia només 4 anys). Els matrimonis eren normalment per conveniència enlloc de per amor.

Impostos:

Tributs senyorials

Delmes eclesiàstics

Rentes reals

Festivitats: Es celebraven a la nit. Durant el dia es treballava.

1.1.2. Els nobles

El grup dels nobles es dividia en dos grups. Per una banda hi havia el braç militar, és a dir, els cavallers, i per l'altra hi havia els Barons (o Ric homens) que eren els que ostentaven el poder polític dels feus i les ciutats.

Activitats que feien:

Si eren temps de guerra, la seva funció era lluitar contra els seus enemics. En el cas de les ciutats, si la guerra s'esdevenia fóra d'aquestes, els cavallers marxaven allà on hi hagués el camp de batalla. En cas de que la guerra fos propera a la ciutat, aleshores venien cavallers de fóra per a lluitar a la ciutat.

Si eren temps de pau, els cavallers participaven en tornejos i caceres.

Vida

Quan un noi naixia dins d'una família noble de cavallers, als 6 anys era enviat al castell d'un altre noble per a treballar com a patges.

Quan arribava als 10-12 anys, era nomenat escuder i començava a entrenar-se per a ser cavaller.

Als 18 anys eren nomenats cavallers i passaven a formar part de l'ordre militar.

1.1.3. El clergue

Activitats que feien

Resaven 8 cops al dia a les següents hores:

00h Maitines

03h Laudes

06h Prima

09 Tercia

12 Sexta

15 Nona

18 Vísperes

21 Completes

Vida

Entraven de petits als convents / monestirs com a novicis. No en sortien i estudiaven religió i cultura.

Un cop superaven l'etapa de novicis, eren nomenats clergues. A la ciutat hi predominaven els Frares, que es dedicaven a la almoina, predicació i vivien als convents.

1.1.4. Serfs

Activitats que feien

Treballaven la terra de sol a sol i donaven una part de les seves collites al noble al qual pertanyien. També li feien feines gratuïtes.

Havien de demanar permís al senyor per a fer qualsevol acció. No podien marxar del domini del senyor.

Vivien en cases on els seus components centrals era un matrimoni, els seus fills i els pares de l'home.

El 70% de la seva alimentació era Pa o similars derivats dels cereals. El 30% era a base de carn o peix, depenent de la zona on vivien.

Aplicaven rotació triennal (per simplificar, la producció era el 66% del total)

Es divertien a les tabernes i als prostíbuls (No sé si es podien permetre entrar a les cases de banys)

Vida

Els nens vivien amb els seus pares i aprenien a cultivar la terra. Les nenes vivien també amb els seus pares i aprenien les feines de la llar de la seva mare.

De joves, les noies eren casades ràpidament i marxaven de casa per a passar a viure amb el seu marit. Els nois no es casaven de tant joves i seguien aprenent a treballar la terra.

De grans, quan els homes ja es podien valdre i tenir cura d'una família, es casaven (amb noies bastant més joves que ells) i formaven una família.

Seguien treballant la terra fins a ser tan vells que no podien fer-ho o fins a la mort.

1.1.5. Vilans: Alta Burgesia

Era el grup format pels grans comerciants i els banquers.

Activitats que feien

Estudiaven a les escoles de nens

Es divertien a les Tabernes i a les cases de banys. També hi havia prostíbuls.

1.1.6. Vilans: Petita Burgesia

Era e grup format pels artesans i persones que treballaven en un taller elaborant productes.

Activitats que feien

Treballaven als seus tallers amb la llum del dia.

Elaboraven productes.

Es divertien a les Tabernes i a les cases de banys. També hi havia prostíbuls.

Vida

Als 7 anys entraven com a aprenents en un taller o eren enviats a servir com a criats a cases d'altres burgesos o noblesa.

Quan arribaven a la majoria d'edat passaven a ser artesans.

Si aconseguien fer una obra mestre, passaven a ser mestres artesans.

1.1.7. Vilans: Jueus

Activitats que feien

Vivien en un barri apart de la ciutat que tancava a les nits.

Es dedicaven a la medicina, orfebreria, banca, tasques administratives i a ser ambaixadors.

Eren MOLT IMPORTANTS a la ciutat de Girona

1.2. La ciutat medieval

1.2.1. Condicions i Estat

Les ciutats tenien unes condicions higièniques molt dolentes. La brossa i residus orgànics s'acumulaven als carrers, que eren de terra. No existien clavegueres ni cap mena de sistema de canalització dels residus humans, que normalment acabaven al carrer!!.

L'únic barri que no estava en condicions insalubres era el barri jueu.

Donat que les cases eren de fusta i que la forma d'il·luminar-se en aquella època es basava en torxes i llànties, era molt freqüent que es produïssin incendis.

1.2.2. Activitat diària

Les persones treballaven normalment amb la llum del sol. Per tant, la jornada laboral era més llarga a l'estiu i més curta a l' hivern. Donat que les ciutats que estudiem es movien dins d'una cultura cristiana, els diumenges no es treballava.

Un cop per setmana es celebrava un mercat on es realitzava la compra de productes que podien venir de fóra de la ciutat.

Els nens de famílies burgeses, nobles o els novicis assistien a l'escola per a adquirir coneixement. Existien tres tipus diferents d'escoles, segons qui les gestionés:

Escoles episcopals (Gestionades pel clergat)

Escoles Urbanes (Gestionades pel govern de la ciutat)

Universitats (Eren lliures)

2. Nutrició

2.1. Taules de Nutrició

2.1.1. Consum Diari per persona

	Pes (Kg)	Kcal	H. C. (g)	Proteïnes	Greixos (g)	Vit. A (mg)	Vit. B1 (mg)	Vit. B2 (mg)	Calci	Fòsfor	Ferro
Homes	30	1960	94	30	34	700	1	1	800	800	10
Dones	30	1960	94	30	34	700	1	1	800	800	10
Nens	30	1960	94	30	34	700	1	1	800	800	10
Homes	50	3266	141	45	51	1000	1	2	1200	1200	18
Dones	45	2939	145	46	53	800	1	1	1200	1200	18

Adolescents	48	3103	143	46	52	900	1	1	1200	1200	18
Homes	70	4572	176	56	64	1000	1	1	800	800	10
Dones	55	3593	138	44	50	800	1	1	800	800	10
Adults	63	4082	157	50	57	900	1	1	800	800	10

2.1.2. Aportació dels diferents Aliments

	Proporció	Kcal	H. C. (g)	Proteïnes(g)	Greixos (g)	Vit. A (mg)	Vit. B1 (mg)	Vit. B2 (mg)	Calci	Fòsfor	Ferro
Avena	20%	400	68	10	6,5	0	0	0	55	400	4
Gluten	10%	380	45	42	2	0	0	0	40	140	0
Blat	20%	360	75	9	2,5	0	0	0	10	290	3
Mandioca	10%	320	80	1,5	0,5	0	0	0	150	100	5
Trigo	40%	340	71	9	1	0	0	0	15	75	2
Blat	100%	358	69,5	11,75	2,45	0	0	0	38	192	2,7
Vaca	20%	200	0	19	13	20	0,05	0,14	9	0	1,8
Vedella	20%	175	0	20	10	20	0,05	0,14	9	0	1,9

Porc	20%	275	0	17	23	0	0,8	0,19	8	160	2,2
Pollastre	20%	170	0	28	10	65	0,08	0,15	11	200	2
Xai	20%	165	0	18	10	0	0	0	0	0	0
Carn	100%	197	0	20,4	13,2	21	0,196	0,124	7,4	72	1,58
Tomàquet	30%	21	0	0	0	900	0,06	0,04	12	26	0,5
Coliflor	20%	28	0	0	0	60	0,11	0,11	27	56	1
Ceba	20%	37	0	0	0	30	0,04	0,04	30	36	0,6
Pastanaga	20%	40	0	0	0	3500	0,06	0,05	40	35	0,9
Fava	10%	105	0	0	0	210	0,25	0,2	29	160	2,3
Verdura	100%	37,8	0	0	0	1009	0,085	0,072	25,9	49,2	0,88

3. Psicologia

A la hora de representar la psicologia de les persones es va fer inevitable conèixer, encara que fos mínimament, una mica de psicologia. Aquest mínim coneixement havia de permetre la representació i utilització del caràcter humà de forma senzilla i ràpida.

Un primer cop d'ull a les possibles representacions del caràcter humà que s'utilitzen va desvetllar que a la psicologia moderna es fan servir complexos sistemes conceptuals per a representar quin pot ser el caràcter dels diferents individus.

Aquests sistemes no m'eren útils en absolut donat que requerien grans estructures per a representar-se i a més a més, la interacció entre dos caràcters diferents es presentava difícil i complexa.

Seguint amb la cerca, no obstant, em vaig topar amb un indicador del caràcter que requeria una quantitat molt petita d'informació per a ser representat i permetia la interacció entre diversos caràcters de forma molt senzilla. Aquesta representació era el MBTI.

3.1. El MBTI (Myers-Briggs Type Indicator)

El MBTI és un indicador de la personalitat de les persones. Segons wikipedia:

El MBTI és un qüestionari psicomètric destinat a mesurar les preferències psicològiques de com percep la gent el món i pren les decisions. Va ser desenvolupat per Katherine Cook Briggs i la seva filla Isabel Myers Briggs a l'any 1962.

Tot i que hi ha molta gent dins de l'àmbit de la psicologia que el critiquen per la seva manca de dades vàlides, el que realment em va cridar l'atenció del MBTI va ser que utilitzava 4 nombres reals per a la representació de tot caràcter. Aquesta representació complia amb els criteris que jo buscava d'una representació fàcil i, a l'hora, la interacció entre diversos caràcters representats mitjançant el sistema MBTI és molt senzilla.

En concret, aquesta representació es basa en quatre dicotomies, o quatre diferents barems sobre els que les persones veiem el mon i prenem les decisions que creiem més convenients. Cadascun d'aquests es denomina per dos paraules descriptives dels dos casos més extrems que es poden donar i s'utilitza un enter entre 0 i 1 per a indicar a quin nivell té la persona en qüestió aquesta dicotomia. Per exemple, la primera dicotomia determina si una persona és

extravertida o introvertida, un 0 en aquest valor indicaria una persona completament extravertida i un 1, una persona completament introvertida.

Les quatre dicotomies són les següents:

Actituds: Extroversion/Introversion: Marcarien si una persona és extravertida, tendirà a actuar ràpidament i de forma espontània mentre que una persona introvertida és més reflexiva i dedicada als seus propis pensament.

Funcions: Sensing / Perceiving: Marcarien si una persona és guia més per les sensacions que li transmet el seu voltant (sensing) o si el que fa és observar el seu entorn en detall i adquirir així informació (perceiving)

Funcions: Thinking / Feeling: Una persona més thinking tendirà a prendre decisions més racionals, basades en la informació de la que disposa. Una persona més feeling prendrà les decisions més basat en sentiments i sensacions.

Forma de vida: Judging / Perception: Una persona més judging tendirà a preferir que les coses estiguin sota control mentre que una persona amb més perception tendirà a no necessitar que tot estigui sota control i de tenir moltes opcions obertes en tot moment.

Amb aquests quatre valors, es podia establir una forma de classificar i representar els caràcters de les persones i permetia establir possibles relacions entre ells.

Aquest mateix estudi mostrava com cada persona tendeix a buscar a la parella valors oposats als seus en les quatre dicotomies o valors molt semblants als seus, mai termes mitjos.

Amb aquestes dades podia començarà establir preferències la hora d'establir les mínimes relacions que regeixen la vida dels membres del simulador.

4. Especificacions i funcionament del SDLPS

El SDLPS és una aplicació que permet la simulació de diagrames SDL de forma nativa.

Aquest software, a més a més, genera una sortida basada en els senyals que es transmetin a través del diagrama SDL.

4.1. Entrada del SDLPS

El SDLPS rep, per una banda, un diagrama SDL especificat en XML. Aquest fitxer XML conté la informació del diagrama SDL codificada de forma 1:1 amb els diagrama original. Les especificacions del XML es poden consultar a la API del SDLPS http://147.83.24.206/~pau/PRSDLPS.html

Un cop es disposa d'aquest XML, el SDLPS el convertirà en un codi C que haurà de ser compilat. En les versions actuals, el SDLPS està preparat per a treballar amb el compilador mingw (http://www.mingv.org) . Si s'ha configurat la localització del compilador, el SDLPS compilarà automàticament el fitxer c general a partir del fitxer c.

S es desitja afegir algun tipus de codi extern al simulador, es pot crear una carpeta a la mateixa ubicació del fitxer XML que s'anomeni SDLCodeExternal i que contingui, entre els que siguin necessaris, un fitxer anomenat SDLCodeExternal.h o s'hi puguin trobar els headers o capçaleres que es criden des de el fitxer XML[32].

Amb els fitxers C, el SDLPS genera un fitxer .DLL que és el que es farà servir, entre d'altres, per a llençar la simulació.

4.2. Simulació

El SDLPS permet la simulació de forma local o través de la xarxa local. El que fa és crear un threat per cada bloc que existeix i enviar-lo a l'equip que creu corresponent. A més a més,

[32] Existeix un tag a l'xml (<task> </task>) que permet introduïr codi C al fitxer xml.

disposa d'una funció anomenada CallReport que es pot invocar des de l' XML i que té l'objectiu d'enviar les dades de la simulació a un equip dedicat exclusivament a la representació de la simulació.

Tot bloc, per tant, s'executa com un únic thread que comparteix informació amb el threat principal i del que en van traient informació quan la necessiten.

4.3. Sortida del SDLPS

Com s'ha comentat abans, el SDLPS permet enviar informació a través de la xarxa cap a un equip que es dediqui exclusivament a la representació. No obstant, el thread principal generarà un fitxer .csv que contindrà informació de totes les senyals que s'envien al llarg de la simulació i que servirà per a debugar el programa.

5. Sandrila

La eina Sandrila SDL es la que es va utilitzar per a la elaboració del diagrames SDL. Aquesta eina consisteix en un plug-in per al MS Visio desenvolupat per l'empresa Sandrila Ltd que permet tant la realització de diagrames SDL mitjançant el MS Visio i a més a més permet l'anàlisi sintàctica dels mateixos per a conèixer-ne la correctesa.

Per a més informació sobre el plug-in Sandrila, per favor, referir-se a http://www.sdl.sandrila.co.uk/

6. Definició de Ciutat

Un dels punts més importants en aquest treball era abstreure quina era la vertadera definició de ciutat. El problema és on es podia trobar quina era la definició verdadera d'una ciutat.

Segons wikipedia, una ciutat és una àrea urbana amb una alta densitat de població i amb un estatus històric, legal o administratiu propi.

Realment, aquesta definició no ens és realment útil per al nostre propòsit, no obstant, es pot treure el primer element que definirà les nostres ciutats: **És una àrea**. I més que això, **és una àrea habitada de forma més densa que la resta del territori**. Per tant, serà important definir un espai i un conjunt de persones que hi habiten i hi realitzen una acció econòmica.

Però amb això no en tenim prou per a definir una ciutat. Cada ciutat funciona de forma diferent i, per tant, calia filar més prim que això.

Si ens centrem en analitzar la activitat econòmica d'una ciutat, podrem veure que cadascuna es dedica a la producció d'una sèrie limitada de bens, aquests bens són, per tant, també part de la definició d'una ciutat i no es podrà entendre ciutat sense parlar de la seva producció de bens. Per tant, serà clau definir quins béns es produeixen a la ciutat.

Lligat als béns, també hi entraria un gran tema en la definició de ciutat: la cultura. La cultura és, com un pot bonament deduir, un element diferenciador clau. La cultura determina els tipus de relacions que s'estableixen entre els membres d'una ciutat, la cultura determina l'alimentació, la cultura en informa, en general, de **com funciona la ciutat** . Segons els edictes que es poden consultar de l'església de l'edat mitjana, una ciutat només era considerada com a tal si tenia una catedral. Això ens hauria de mostrar com de profunda és la relació entre la cultura (en aquest cas, la religió) i la definició de ciutat.

En resum, tenim tres elements que en defineixen la ciutat:

- Àrea
- Població
- Cultura

Són aquests tres elements els que defineixen la ciutat i els que hauran de ser tinguts en compte si es vol poder parametritzar una ciutat de forma correcta.

7. Model Lemuria.xml

```
<?XML version="1.0" encoding="utf-8"?>

<!--
AUTOR: Manel Rello Saltor manelrello.multiply.com
DATA DE CREACIó: 11/08/2008
CONTINGUT: Lemuria.xml Definició del simulador de ciutats

Darrera modificació: 24/11/2008

<shiporder orderid="889923"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="shiporder.xsd">
-->

<system id="0" name="Lemuria" version="1.0" implementation="" IP="" portRead=""
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="SDLPS.xsd">
<channels>
        <channel name="Monthly_Enviroment_Channel" start="block_enviroment"
end="block_household" dual="yes">
                <event name="monthly_env"></event>
```

```
        </channel>
        <channel name="Production_Household" start="block_familiar_unit"
end="block_household" dual="yes">
                <event name="start_production"></event>
                <event name="resources"></event>
                <event name="buy_household"></event>
                <event name="sell_household"></event>
        </channel>

        <channel name="Env_Conditions_Channel" start="block_familiar_unit"
end="block_enviroment" dual="yes">
                <event name="monthly_env_UF"></event>
                <event name="monthly_env_UF2"></event>
        </channel>

        <channel name="Couple_Channel" start="block_familiar_unit"
end="block_couple_market" dual="yes">
                <event name="add_marriageable"></event>
                <event name="marriage"></event>
                <event name="remove_FU"></event>
        </channel>
        <channel name="CM_Enviroment" start="block_enviroment"
end="block_couple_market" dual="yes">
                <event name="CM_month"></event>
        </channel>
```

```
        <channel name="MK_Enviroment" start="block_enviroment" end="block_market" dual="yes">
                <event name="MK_month"></event>
        </channel>
        <channel name="MK_UF" start="block_market" end="block_familiar_unit" dual="yes">
                <event name="buy_resources"></event>
                <event name="sell_resources"></event>
        </channel>
</channels>

<block id="1" name="block_familiar_unit" implementation="" IP="" portRead="">
        <channels>
                <channel name="IO_harvest_process" start="block_familiar_unit" end="process_harvest_process" dual="yes">
                        <event name="buy_resources"></event>
                        <event name="sell_resources"></event>
                        <event name="resource_request"></event>
                        <event name="resource_obtained"></event>
                        <event name="monthly_env_UF"></event>
                        <event name="start_production"></event>
                        <event name="resources"></event>
                        <event name="buy_household"></event>
                        <event name="buy_resources"></event>
                        <event name="sell_resources"></event>
                </channel>
```

```
            <channel name="IO_life_process" start="block_familiar_unit"
end="process_life_process" dual="yes">
                <event name="monthly_env_UF2"></event>
                <event name="add_marriageable"></event>
            </channel>

        </channels>

        <process id="1" name="process_harvest_process" implementation="" IP=""
portRead="">
            <DCLS>

                <DCL name="UF_id" type="int" value="0"></DCL>

                <DCL name="T" type="double" value="0"></DCL>
                <DCL name="b" type="int" value=""></DCL>
                <DCL name="id" type="int" value="0"></DCL>

                <DCL name="inteligence" type="int" value="100"></DCL>



                <DCL name="UF_temperature" type="int" value=""></DCL>
                <DCL name="UF_wind" type="int" value=""></DCL>
                <DCL name="UF_rain" type="int" value=""></DCL>
```

```
<DCL name="UF_month" type="int" value="0"></DCL>

<DCL name="UF_product" type="int" value="0"></DCL>
<DCL name="UF_qtty" type="int" value="0"></DCL>

<DCL name="UF_action" type="int" value="0"></DCL>

<DCL name="UF_param1" type="int" value="0"></DCL>
<DCL name="UF_param2" type="int" value="0"></DCL>
<DCL name="UF_param3" type="int" value="0"></DCL>
<DCL name="UF_param4" type="int" value="0"></DCL>
<DCL name="UF_param5" type="int" value="0"></DCL>
<DCL name="UF_action_type" type="int" value="0"></DCL>

</DCLS>
<procedures>
<procedure id="1" name="modify_weather_knowledge" implementation="">
<body>
<task id="1" name="">
uf_modify_weather_knowledge(UF_id, UF_month, UF_temperature, UF_wind, UF_rain);
</task>
</body>
</procedure>
```

```
<procedure id="2" name="modify_res_knowledge" implementation="">
<body>
<task id="2" name="">
uf_modify_resources_knowledge(UF_id);
uf_modify_future_resources_knowledge(UF_id);
</task>
</body>
</procedure>
<procedure id="3" name="do_action" implementation="">
<params>

</params>
<body>
<task id="1" name="">
UF_action = uf_get_action(UF_id, UF_2month);
</task>
</body>
</procedure>
<procedure id="4" name="update_action" implementation="">
<params>

</params>
<body>
<task id="1" name="">
```

```
                    uf_get_action_params(UF_id, _POINT_ UF_action_type,
_POINT_ UF_param1, _POINT_ UF_param2, _POINT_ UF_param3, _POINT_ UF_param4,
_POINT_ UF_param5);
                    </task>
                    </body>
                    </procedure>
            </procedures>
            <start>
                <setstate id="1" name="WAITING"></setstate>
            </start>
            <state name="WAITING">


                <input id="6" name="resources"></input>
                <output id="8" name="information_processed" self="yes" via="">
                    <param name="delay" value="0"/>
                    <param name="priority" value="1"/>
                    <userparam name="UF_id" value="UF_id"/>
                </output>
                <task id="10" name="">
                uf_add_stock(UF_id, UF_product, UF_qtty);
                </task>
                <procedurecall id="11"
name="modify_res_knowledge"></procedurecall>
                <setstate id="12" name="WAITING"></setstate>
```

```
<input id="1" name="monthly_env_UF"></input>
<output id="3" name="information_processed" self="yes" via="">
    <param name="delay" value="0"/>
    <param name="priority" value="1"/>
    <userparam name="UF_id" value="UF_id"/>
</output>
<procedurecall id="4" name="modify_weather_knowledge"></procedurecall>

<procedurecall id="2004" name="do_action">
</procedurecall>

<decision id="2006" name="SR1" iftrue="2007" iffalse="10007">UF_action>0</decision>

<task id="2007" name="">uf_get_action_params(UF_id, _POINT_ UF_action_type, _POINT_ UF_param1, _POINT_ UF_param2, _POINT_ UF_param3, _POINT_ UF_param4, _POINT_ UF_param5);</task>
<decision id="2008" name="CHECK_COND" iftrue="2009" iffalse="10007">UF_action_type>=0</decision>
<decision id="2009" name="BUY_HH" iftrue="1009" iffalse="1010">UF_action_type==0</decision>
    <output id="1009" name="buy_household" self="" via="IO_harvest_process">
        <param name="delay" value="1"/>
        <param name="priority" value="0"/>
```

```
                    <userparam name="posX" value="UF_param1"/>
                    <userparam name="posY" value="UF_param2"/>
                    <userparam name="propietari" value="UF_param3"/>

                </output>

            <decision id="1010" name="START_HH" iftrue="1011"
iffalse="1012">UF_action_type==1</decision>
                <output id="1011" name="start_production" self=""
via="IO_harvest_process">
                    <param name="delay" value="2"/>
                    <param name="priority" value="0"/>
                    <userparam name="posX" value="UF_param1"/>
                    <userparam name="posY" value="UF_param2"/>
                    <userparam name="HH_cultiu" value="UF_param3"/>
                    <userparam name="HH_persones"
value="UF_param4"/>
                    <userparam name="HH_experiencia"
value="UF_param5"/>
                </output>

                <decision id="1012" name="SELL_HH" iftrue="1013"
iffalse="1014">UF_action_type==2</decision>
                <output id="1013" name="sell_household" self=""
via="IO_harvest_process">
                    <param name="delay" value="0"/>
```

```
                    <param name="priority" value="0"/>
                    <userparam name="posX" value="UF_param1"/>
                    <userparam name="posY" value="UF_param2"/>

                </output>

                <decision id="1014" name="BUY_RES" iftrue="1015"
iffalse="1016">UF_action_type==3</decision>
                <output id="1015" name="buy_resources" self=""
via="IO_harvest_process">
                    <param name="delay" value="0"/>
                    <param name="priority" value="0"/>
                    <userparam name="MK_UF_id" value="UF_param1"/>
                    <userparam name="MK_Type" value="UF_param4"/>
                    <userparam name="MK_Res" value="UF_param2"/>
                    <userparam name="MK_Qtty" value="UF_param3"/>

                </output>

                <decision id="1016" name="SELL_RES" iftrue="1017"
iffalse="10006">UF_action_type==4</decision>
                <output id="1017" name="sell_resources" self=""
via="IO_harvest_process">
                    <param name="delay" value="0"/>
                    <param name="priority" value="0"/>
                    <userparam name="MK_UF_id" value="UF_param1"/>
```

```
                                <userparam name="MK_Type" value="UF_param4"/>
                                <userparam name="MK_Res" value="UF_param2"/>
                                <userparam name="MK_Qtty" value="UF_param3"/>

                            </output>

                    <decision id="10006" name="Bucle" iftrue="2007"
iffalse="2007">1==1</decision>

                    <output id="10007" name="action_executed" self="yes" via="">
                            <param name="delay" value="0"/>
                            <param name="priority" value="1"/>
                            <userparam name="UF_action" value="UF_action"/>
                    </output>

                    <setstate id="10008" name="WAITING"></setstate>
        </state>
        </process>

        <process id="2" name="process_life_process" implementation="" IP=""
portRead="">

        <DCLS>
```

```
<DCL name="UF_2Dead" type="int" value="0"></DCL>
<DCL name="UF_2Birth" type="int" value="0"></DCL>
<DCL name="UF_2widow" type="int" value="0"></DCL>
<DCL name="UF_2orphan" type="int" value="0"></DCL>

<DCL name="UF_2month" type="int" value="0"></DCL>
<DCL name="UF_2newchild" type="int" value="0"></DCL>

<DCL name="UF_2id" type="int" value="0"></DCL>
<DCL name="UF_2marriageable" type="int" value="0"></DCL>

<DCL name="UF_filePath" type="char*" value=""></DCL>

</DCLS>
<procedures>
    <procedure id="1" name="computeHealth" implementation="">
        <body>
        <task id="1" name="">
        if (UF_2month == 0){
            debug_msg("Compute Health\n");
            uf_compute_health(UF_2id);
            debug_msg("Compute Death\n");
            UF_2Dead = uf_compute_death(UF_2id);
            debug_msg("Compute Birth\n");
            UF_2Birth = uf_compute_birth(UF_2id);
```

```
        }
        else {
            UF_2Dead = -1;
            UF_2Birth = 0;
        }
        </task>
        </body>
    </procedure>

    <procedure id="2" name="computeParental" implementation="">
        <body>
        <task id="2" name="">member_death(UF_2Dead); </task>
        <task id="3" name="">UF_2id = member_get_UF(UF_2Dead); </task>
        <task id="4" name="">UF_2widow = uf_get_widow(UF_2id);</task>
        <task id="5" name="">UF_2orphan = uf_is_orphan(UF_2id);</task>

        </body>
    </procedure>
    <procedure id="3" name="addChild" implementation="">
        <body>
        <task id="3" name="">
        UF_2newchild=uf_add_child(UF_2id);
        </task>
        </body>
    </procedure>
```

```
            <procedure id="4" name="advance_age" implementation="">
                <params>
                    <param name="person" type="int" defvalue="" ref="yes"></param>
                </params>
                <body>
                <task id="4" name="">member_advance_age(person);</task>
                </body>
            </procedure>

            <procedure id="5" name="OpenFile" implementation="CSDLOperationProcedureCallOpenFile">
                <params>
                    <param name="Path" type="char*" defvalue="" ref="yes"></param>
                </params>
            </procedure>
        </procedures>

        <start>

            <task id="1" name="">init_lemuria_part1();</task>
            <procedurecall id="4" name="OpenFile">
                <param name="Path" value="UF_filePath"/>
            </procedurecall>
```

```
            <task id="3" name="">readModel(Path);</task>
            <task id="4" name="">init_lemuria_part2();</task>
            <setstate id="100" name="ALIFE"></setstate>
        </start>
        <state name="ALIFE">
        <input id="1" name="monthly_env_UF2"></input>
        <!-- Falta mirar si tenim algun major d'edat per a poder-lo enviar a casar-se...-->
        <procedurecall id="4" name="computeHealth"></procedurecall>
        <decision id="5" name="" iftrue="1001" iffalse="2001">UF_2Dead!=-1</decision>
        <output id="1001" name="member_death" self="yes" via="">
                        <param name="delay" value="0"/>
                        <param name="priority" value="0"/>
                        <userparam name="UF_2Dead" value="UF_2Dead"/>
                        <userparam name="UF_2id" value="UF_2id"/>
        </output>
        <decision id="2001" name="" iftrue="3001"
iffalse="4001">UF_2Birth!=0</decision>
        <output id="3001" name="member_birth" self="yes" via="">
                        <param name="delay" value="0"/>
                        <param name="priority" value="0"/>
                        <userparam name="UF_2id" value="UF_2id"/>
        </output>

        <setstate id="4001" name="ALIFE"></setstate>
```

```
<input id="6" name="marriage"></input>
<!-- Mirar si alguna cosa s'ha de tocar-->
<setstate id="9" name="ALIFE"></setstate>

<input id="10" name="member_death"></input>
<procedurecall id="11" name="computeParental"></procedurecall>
<decision id="12" name="SR1" iftrue="6002" iffalse="5001">UF_2widow>-1</decision>

<decision id="6002" name="SR1" iftrue="6101" iffalse="6201">UF_2orphan==1</decision>
<!--MANEL: Modificar, no és l'estat el que serà DEAD sino que hem d'esborrar la família en si...-->
<setstate id="6101" name="ALIFE"></setstate>

<output id="6201" name="add_marriageable" self="" via="IO_life_process">
                    <param name="delay" value="0"/>
                    <param name="priority" value="0"/>

                    <userparam name="member1" value="UF_2widow"/>
</output>

<setstate id="5001" name="ALIFE"></setstate>
```

```
        <input id="14" name="member_birth"></input>

        <procedurecall id="15" name="addChild"></procedurecall>

        <output id="6201" name="add_marriageable" self="" via="IO_life_process">
                        <param name="delay" value="0"/>
                        <param name="priority" value="0"/>

                        <userparam name="member1" value="UF_2newchild"/>
        </output>

        <setstate id="19" name="ALIFE"></setstate>




        </state>
        <state name="ALIFE">

        </state>
        </process>
</block>

<block id="2" name="block_market" implementation="" IP="" portRead=""> <!-- Complet --
>
```

```
<channels>
        <channel name="IO_market_process" start="process_market_process" end="block_market" dual="yes">
            <event name="sell_resources"></event>
            <event name="buy_resources"></event>
            <event name="MK_month"></event>
        </channel>

</channels>
<process id="1" name="process_market_process" implementation="" IP="" portRead="">
        <DCLS>
            <DCL name="MK_UF_id" type="int" value=""></DCL>
            <DCL name="MK_Type" type="int" value=""></DCL>
            <DCL name="MK_Res" type="int" value=""></DCL>
            <DCL name="MK_Qtty" type="int" value=""></DCL>

            <DCL name="MK_month" type="int" value=""></DCL>
        </DCLS>
        <procedures>
            <procedure id="1" name="evolve_princing" implementation="">
                <params>

                </params>
                <body>
```

```
                    <task id="1" name="">market_evolve_prices(MK_month);</task>
                    </body>
            </procedure>
                <procedure id="2" name="compute_shopping" implementation="">
              <params>

                    </params>
                    <body>
                        <task id="1" name="">market_add_order(MK_UF_id, MK_Type, MK_Res, MK_Qtty);</task>
                    </body>
            </procedure>
             </procedures>
             <start>
                <setstate id="1" name="OPEN"></setstate>
             </start>
             <state name="OPEN">
                <input id="1" name="buy_resources"></input>
                <procedurecall id="2" name="compute_shopping"></procedurecall>
                <output id="9997" name="res_bought" self="yes" via="">
                    <param name="delay" value="0"/>
                    <param name="priority" value="0"/>
                </output>
                <setstate id="3" name="OPEN"></setstate>
```

```
                    <input id="4" name="sell_resources"></input>
                    <procedurecall id="5" name="compute_shopping"></procedurecall>
                    <output id="9998" name="res_sold" self="yes" via="">
                        <param name="delay" value="0"/>
                        <param name="priority" value="0"/>
                    </output>
                    <setstate id="6" name="OPEN"></setstate>

                    <input id="7" name="MK_month"></input>
                    <procedurecall id="8" name="evolve_princing"></procedurecall>
                    <output id="9999" name="prices_evolved" self="yes" via="">
                        <param name="delay" value="0"/>
                        <param name="priority" value="0"/>
                    </output>
                    <setstate id="9" name="OPEN"></setstate>
                </state>
            </process>


</block>


<block id="3" name="block_couple_market" implementation="" IP="" portRead=""> <!--
Complet -->
            <channels>
```

```
            <channel name="IO_couple_process" start="block_couple_market"
end="process_couple_process" dual="yes">
                <event name="add_marriageable"></event>
                <event name="marriage"></event>
                <event name="remove_FU"></event>
                <event name="CM_month"></event>
            </channel>



    </channels>

    <process id="1" name="process_couple_process" implementation="" IP=""
portRead="">
    <DCLS>
            <DCL name="member1" type="int" value=""></DCL>
            <DCL name="member2" type="int" value=""></DCL>
            <DCL name="found" type="int" value=""></DCL>
            <DCL name="parent1" type="int" value=""></DCL>
            <DCL name="parent2" type="int" value=""></DCL>
            <DCL name="UF_id1" type="int" value=""></DCL>
            <DCL name="UF_id2" type="int" value=""></DCL>
            <DCL name="cm_m1" type="int" value="0" ></DCL>
            <DCL name="cm_m2" type="int" value="0" ></DCL>
    </DCLS>
```

```
<procedures>
        <procedure id="1" name="search_couple" implementation="">
            <params>
                <param name="pmember1" type="int" defvalue="" ref="no"></param>
                <param name="pmember2" type="int" defvalue="" ref="yes"></param>
                <param name="pfound" type="int" defvalue="" ref="yes"></param>
            </params>
            <body>
            <task id="1" name="">pfound = search_couple(pmember1, _POINT_ pmember2, 1);</task>
            </body>
        </procedure>
    </procedures>
<start>
    <setstate id="1" name="ACTIVE"></setstate>
</start>
<state name="ACTIVE">
    <input id="10" name="add_marriageable"></input>
    <procedurecall id="2" name="search_couple">
        <param name="pmember1" value="member1"/>
        <param name="pmember2" value="member2"/>
        <param name="pfound" value="found"/>
```

```
</procedurecall>
<decision id="3" name="SR1" iftrue="4" iffalse="7">found</decision>
<output id="4" name="engage" self="yes" via="">
        <param name="delay" value="0"/>
        <param name="priority" value="0"/>
        <userparam name="member1" value="member1"/>
        <userparam name="member2" value="member2"/>
</output>

<setstate id="7" name="ACTIVE"></setstate>

<input id="3001" name="CM_month"></input>
<task id="3002" name="">found=search_couples(_POINT_ cm_m1, _POINT_ cm_m2);</task>
<decision id="3003" name="NOVES_PARELLES" iftrue="3101" iffalse="3004">found</decision>
<output id="3101" name="engage" self="yes" via="">
        <param name="delay" value="0"/>
        <param name="priority" value="0"/>
        <userparam name="member1" value="cm_m1"/>
        <userparam name="member2" value="cm_m2"/>
</output>
<decision id="3102" name="BUCLE" iftrue="3002" iffalse="3002">1==1</decision>
<setstate id="3004" name="ACTIVE"/>
```

```
            <input id="10" name="engage"></input>
            <task id="11" name="">process_mariage(_POINT_ member1, _POINT_
member2);</task>
            <task id="12" name="">parent1=is_parent(member1);</task>
            <task id="13" name="">parent2=is_parent(member2);</task>
            <setstate id="2217" name="ACTIVE"/>

        </state>
        </process>

</block>

<block id="4" name="block_household" implementation="" IP="" portRead="">
  <layers dim="100,100">
   <layer name="" data="">
    <nucleous body="N">{[0]}</nucleous>
    <neighborhood body="N">{[1,0],[1,1],[0,1],[-1,0],[-1,-1],[0,-1],[1,-1],[-
1,1]}</neighborhood>
   </layer>
  </layers>
  <channels>
```

```
<channel name="IO_household_process" start="cell_household" end="block_household" dual="yes">
<event name="monthly_env"></event>
<event name="buy_household"></event>
<event name="sell_household"></event>
<event name="start_production"></event>
<event name="resources"></event>
</channel>
</channels>
<!--Els blocs que representen les cel·les contenen la funció d'evolució-->
<block id="1" name="cell_household" implementation="" IP="" portRead="">
<channels>
<channel name="IO_household_process_cell" start="process_household_process" end="cell_household" dual="yes">
<event name="monthly_env"></event>
<event name="buy_household"></event>
<event name="sell_household"></event>
<event name="start_production"></event>
<event name="resources"></event>
</channel>
</channels>

<process id="1" name="process_household_process" implementation="" IP="" portRead="">
<DCLS>
```

```
<DCL name="posX" type="int" value="" ></DCL>
<DCL name="posY" type="int" value="" ></DCL>
<DCL name="Coord" type="int" value="" ></DCL>
<DCL name="HH_id" type="int" value="" ></DCL>

<DCL name="HH_state" type="int" value="6" ></DCL>

<DCL name="currmonth" type="int" value="0"></DCL>
<DCL name="hh_product" type="int" value="0"></DCL>
<DCL name="hh_qtty" type="int" value="0"></DCL>

<DCL name="duration" type="int" value="12" ></DCL>
<DCL name="propietari" type="int" value="12" ></DCL>

<DCL name="temperature" type="int" value="0"></DCL>
<DCL name="wind" type="int" value="0"></DCL>
<DCL name="rain" type="int" value="0"></DCL>
<DCL name="HH_cultiu" type="int" value="0"></DCL>
<DCL name="HH_persones" type="int" value="0"></DCL>
<DCL name="HH_experiencia" type="int" value="0"></DCL>

</DCLS>
```

```
<procedures>
 <procedure id="1" name="computeResources" implementation="">
  <params>

  </params>

  <body>
   <task id="1" name="">

                    hh_advance_month(HH_id, temperature, rain, wind,
currmonth);
                    hh_gather_resources(HH_id, currmonth, _POINT_
hh_product, _POINT_ hh_qtty);

   </task>
  </body>
 </procedure>

        <procedure id="2" name="GetmncaState"
implementation="CSDLOperationProcedureCallGetmncaState">
        <params>
            <param name="layer" type="int" defvalue="" ref="yes"></param>
            <param name="coord" type="int" defvalue="" ref="yes"></param>
            <param name="stateVar" type="int" defvalue="" ref="yes"></param>
        </params>
        </procedure>
```

```
</procedures>

    <start>
        <task id="1" name="">/*hh_initialize();*/</task>
  <setstate id="2" name="CELL"/>
</start>

        <state name="CELL">
        <!-- Buy Household. Ha d'estar en l'estat d'Empty -->
        <input id="11001" name="buy_household"></input>
         <task id="11002" name="">HH_id=posX*100+posY;</task>

         <task id="11005" name="">HH_state=hh_get_state(HH_id);</task>

         <decision id="11006" name="" iftrue="11008"
iffalse="11011">HH_state==EMPTY</decision>
         <task id ="11008" name="">hh_set_propietari(HH_id, propietari);</task>
         <task id="11010" name="">hh_set_state(HH_id, 2);</task>
         <task id="997" name="">HH_state=hh_get_state(HH_id);</task>

<setstate id="11011" name="CELL"/>
```

```
<!-- Start production. Ha d'estar en READY-->
<input id="21009" name="start_production"></input>
<task id="21010" name="">HH_id=posX*100+posY;</task>
<task id="21012" name="">hh_sembra(HH_id, HH_cultiu, HH_persones, HH_experiencia);</task>
<task id="21013" name="">HH_state=hh_get_state(HH_id);</task>

<decision id="21014" name="" iftrue="21017" iffalse="21018">HH_state==READY</decision>
<task id="21017" name="">hh_set_state(HH_id, PRODUCING);</task>
<setstate id="21018" name="CELL"/>

<!-- Aquest dona guerra perquè ens arriba una senyal que s'ha de convertir en moltes. Per sort, si és la genèrica, la que ve del bloc d'Enviroment, ve amb un -1 al HH_id -->
<input id="31000" name="monthly_env"></input>
<task id="31001" name="">HH_id=0;</task>
<decision id="31002" name="" iftrue="31006" iffalse="31003">HH_id==1000</decision>
<decision id="31003" name="" iftrue="32002" iffalse="31004">hh_get_state(HH_id) == PRODUCING</decision>
<task id="32002" name="">propietari = hh_get_propietari(HH_id);</task>
<procedurecall id="32003" name="computeResources">

</procedurecall>
```

```
<output id="32004" name="resources" self="" via="IO_household_process_cell">
<param name="delay" value="0"></param>
<param name="priority" value="0"></param>
<userparam name="UF_product" value="hh_product"></userparam>
<userparam name="UF_qtty" value="hh_qtty"></userparam>
<userparam name="UF_id" value="propietari"></userparam>

</output>

<decision id="32005" name="" iftrue="32006" iffalse="32105">hh_get_sembrat(HH_id)</decision>
<task id="32006" name="">hh_set_state(HH_id, PRODUCING);</task>
<decision id="32007" name="" iftrue="31004" iffalse="31004">1==1</decision>
<task id="32105" name="">hh_set_state(HH_id, READY);</task>

<task id="31004" name="">HH_id++;</task>
<decision id="31005" name="" iftrue="31002" iffalse="31002">1==1</decision>

<setstate id = "31006" name="CELL"/>
</state>
```

```
        </process>


      </block>
  </block>



<block id="5" name="block_enviroment" implementation="" IP="" portRead="">
   <channels>
    <channel name="monthly_enviroment" start="process_monthly_process"
end="process_enviroment_process" dual="yes">
      <event name="month"></event>
    </channel>
    <channel name="IO_enviroment_process" start="process_enviroment_process"
end="block_enviroment" dual="yes">
      <event name="env_conditions"></event>
      <event name="monthly_env"></event>
              <event name="monthly_env_UF"></event>
              <event name="monthly_env_UF2"></event>
              <event name="MK_month"></event>
   </channel>
       <channel name="IO_monthly" start="process_monthly_process"
end="block_enviroment" dual="yes">
      <event name="CM_month"></event>


   </channel>
  </channels>
```

```
<process id="1" name="process_enviroment_process" implementation="" IP="" portRead="">
 <DCLS>
  <DCL name="m_temperature" type="int" value=""></DCL>
  <DCL name="m_wind" type="int" value=""></DCL>
  <DCL name="m_rain" type="int" value=""></DCL>
  <DCL name="mes_actual" type="int" value="0"></DCL>
        <DCL name="any_actual" type="int" value="0"></DCL>
        <DCL name="Env_UF_id" type="int" value="0"></DCL>
</DCLS>

<procedures>
 <procedure id="2" name="MonthClimate" implementation="">
  <params>

  </params>

  <body>
   <task id="1" name="">
    if (mes_actual == 0){
```

```
            setYearDiffTemp(randomInt(-3, 3));
            setYearDiffRain(randomInt(-50, 50));
                        members_advance_age();

                        debug_all(any_actual);
                        any_actual++;
        }
        m_temperature = getTemperature(currmonth)+randomInt(-2, 2);
        m_rain = getRain(currmonth)+randomInt(-20, 20);
        m_wind = randomInt(0, 5);

      </task>
    </body>
  </procedure>
</procedures>

<start>
  <task id="20" name="">
    initializeTemperature();
    initializeRain();
  </task>
  <setstate id="1" name="WORKING_E"></setstate>
</start>
<state name="WORKING_E">
  <input  id="2" name="month"></input>
```

```
<procedurecall id="4" name="MonthClimate">
</procedurecall>
<task id="5" name="">mes_actual = (mes_actual + 1)%12;</task>
<output id="6" name="monthly_env" self="" via="IO_enviroment_process">
 <param name="delay" value="0"/>
 <param name="priority" value="0"/>
 <userparam name="temperature" value="m_temperature"/>
 <userparam name="wind" value="m_wind"/>
 <userparam name="rain" value="m_rain"/>
 <userparam name="currmonth" value="mes_actual"/>
        <userparam name="HH_id" value="-1"/>
</output>
        <!--Bucle per a totes les UFs-->
        <task id="10003" name="">Env_UF_id = 0;</task>
        <decision id="10004" name="Init_Bucle" iftrue="7" iffalse="9">Env_UF_id
!= num_UFs</decision>
        <output id="7" name="monthly_env_UF" self=""
via="IO_enviroment_process">
 <param name="delay" value="0"/>
 <param name="priority" value="0"/>
 <userparam name="UF_temperature" value="m_temperature"/>
 <userparam name="UF_wind" value="m_wind"/>
 <userparam name="UF_rain" value="m_rain"/>
 <userparam name="UF_month" value="mes_actual"/>
        <userparam name="UF_id" value="Env_UF_id"/>
</output>
```

```
<output id="8" name="monthly_env_UF2" self="" via="IO_enviroment_process">
<param name="delay" value="0"/>
<param name="priority" value="0"/>
<userparam name="UF_2month" value="mes_actual"/>
<userparam name="UF_2id" value="Env_UF_id"/>
</output>

<task id="10005" name="">Env_UF_id++;</task>
<decision id="10006" name="Bucle" iftrue="10004" iffalse="10004">1==1</decision>

<output id="9" name="MK_month" self="" via="IO_monthly">
<param name="delay" value="0"/>
<param name="priority" value="0"/>
<userparam name="MK_month" value="mes_actual"/>
</output>

<!-- Final de Bucle -->
<setstate id="10" name="WORKING_E"></setstate>
</state>
</process>

<process id="3" name="process_monthly_process" implementation="" IP="" portRead="">
```

```
<start>
 <output id="2" name="month" self="yes" via="">
  <param name="delay" value="0"/>
  <param name="priority" value="0"/>
 </output>
 <setstate id="1" name="WORKING_M"></setstate>
</start>
<state name="WORKING_M">
 <input id="3" name="month"></input>
 <output id="4" name="month" self="yes" via="">
  <param name="delay" value="30"/>
  <param name="priority" value="0"/>
 </output>
 <output id="5" name="month" self="" via="monthly_enviroment">
  <param name="delay" value="30"/>
  <param name="priority" value="0"/>
 </output>
        <output id="5" name="CM_month" self="" via="IO_monthly">
 <param name="delay" value="30"/>
 <param name="priority" value="0"/>
 </output>

 <setstate id="6" name="WORKING_M"></setstate>
</state>
</process>
```

```
</block>
</system>
```